2006-09

Low-power fault tolerance for spacecraft FPGA-based numerical computing

Snodgrass, Joshua D.

Monterey, California.: Naval Postgraduate School, 2006.

http://hdl.handle.net/10945/10160

NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

DISSERTATION

LOW-POWER FAULT TOLERANCE FOR SPACECRAFT FPGA-BASED NUMERICAL COMPUTING

by

Joshua D. Snodgrass

September 2006

Dissertation Supervisor: Herschel H. Loomis, Jr.

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.			
1. AGENCY USE ONLY ***(Leave blank)***	**2. REPORT DATE** September 2006	**3. REPORT TYPE AND DATES COVERED** Doctoral Dissertation	
4. TITLE AND SUBTITLE: Low-Power Fault Tolerance for Spacecraft FPGA-Based Numerical Computing			**5. FUNDING NUMBERS**
6. AUTHOR(S) Joshua D. Snodgrass			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited			**12b. DISTRIBUTION CODE**

13. ABSTRACT ***(maximum 200 words)***

Fault tolerance is explored for spacecraft computers employing Field-Programmable Gate Arrays (FPGAs). Techniques are investigated for tolerating Single Event Upsets (SEUs) caused by radiation in the space environment. A new architectural approach is proposed for achieving SEU tolerance that minimizes power and size overhead costs by reducing the precision with which error checking is done. This Reduced Precision Redundancy (RPR) approach is compared to the traditional Triple Modular Redundancy (TMR) method. A methodology is presented for quantifying the costs and benefits of various performance factors, and thereby determining optimal design solutions. This methodology considers reliability as a performance factor that can be traded-off against factors such as power, size and speed.

An SEU simulation system is developed for studying the effect of SEUs on actual FPGA circuits. Live proton radiation testing and computer-controlled fault injection simulations demonstrate the effectiveness of RPR and TMR. Computer simulations of power usage demonstrate the savings achieved with RPR. RPR is as reliable as TMR while requiring 1/3 to 1/2 as much power. The effect of imprecise computations that may be produced by an RPR system is studied. An image processing application illustrates the type of problems for which RPR can be applied effectively.

14. SUBJECT TERMS FPGA, computer, fault tolerance, power, single event upset, reliability, spacecraft, space radiation			**15. NUMBER OF PAGES** 247
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UL

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

LOW-POWER FAULT TOLERANCE FOR SPACECRAFT FPGA-BASED NUMERICAL COMPUTING

Joshua D. Snodgrass
Major, United States Air Force
B.S., Stanford University, 1995
M.S., Stanford University, 1996

Submitted in partial fulfillment of the
requirements for the degree of

DOCTOR OF PHILOSOPHY IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
September 2006

Author: _____
Joshua D. Snodgrass

Approved by:

_____
Herschel H. Loomis, Jr., Professor
Electrical and Computer Engr.
Dissertation Supervisor

_____
Jon T. Butler, Professor
Electrical and Computer Engr.

_____
Douglas J. Fouts, Professor
Electrical and Computer Engr.

_____
Sherif Michael, Professor
Electrical and Computer Engr.

_____
Alan A. Ross, Visiting Professor
Space Systems Academic Group

Approved by: _____
Jeffrey B. Knorr, Chair, Department of Electrical and Computer Engr.

Approved by: _____
Julie Filizetti, Associate Provost for Academic Affairs

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Fault tolerance is explored for spacecraft computers employing Field-Programmable Gate Arrays (FPGAs). Techniques are investigated for tolerating Single Event Upsets (SEUs) caused by radiation in the space environment. A new architectural approach is proposed for achieving SEU tolerance that minimizes power and size overhead costs by reducing the precision with which error checking is done. This Reduced Precision Redundancy (RPR) approach is compared to the traditional Triple Modular Redundancy (TMR) method. A methodology is presented for quantifying the costs and benefits of various performance factors, and thereby determining optimal design solutions. This methodology considers reliability as a performance factor that can be traded-off against factors such as power, size and speed.

An SEU simulation system is developed for studying the effect of SEUs on actual FPGA circuits. Live proton radiation testing and computer-controlled fault injection simulations demonstrate the effectiveness of RPR and TMR. Computer simulations of power usage demonstrate the savings achieved with RPR. RPR is as reliable as TMR while requiring 1/3 to 1/2 as much power. The effect of imprecise computations that may be produced by an RPR system is studied. An image processing application illustrates the type of problems for which RPR can be applied effectively.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

I. INTRODUCTION 1
A. OVERVIEW 1
B. BACKGROUND 1
1. Computing In the Space Environment 1
2. Single Event Upset (SEU) 3
3. Field-Programmable Gate Array (FPGA) 5
4. Power Consumption 7
C. MOTIVATION 8
D. CONTRIBUTIONS 9
E. DISSERTATION ORGANIZATION 10

II. FAULT-TOLERANT DESIGN CONCEPTS 11
A. FAULT TOLERANCE FOR FPGAS 11
B. PRINCIPLES OF FAULT TOLERANCE 12
1. Fault/Error Detection vs. Correction 12
2. Concurrent Error Detection and Checkpointing 12
3. Configuration Scrubbing 14
4. Redundancy 15
a. Configuring Redundant Components 16
b. Selective Redundancy 17
5. Complexity/Confidence Trade-offs 18
C. ERROR CODING TECHNIQUES 20
D. TRIPLE MODULAR REDUNDANCY (TMR) 21
E. REDUCED PRECISION REDUNDANCY (RPR) 23
1. Background 23
2. Architecture Description 25
3. Applying RPR to Computational Problems 27
a. Class A Problems 27
b. Class B Problems 29
c. Approximate Solutions 30
d. RPR Suitability 34
e. Examples 36
f. Applying RPR to non-FPGA Systems 38
5. Flexible Precision Computation 39
F. VOTER ISSUES 40
G. SUMMARY 42

III. POWER SAVINGS TECHNIQUES 45
A. POWER EFFICIENCY FOR FPGA DESIGNS 45
B. BACKGROUND 45
1. Relative Contributions of Static and Dynamic Power 47
2. Reducing Static Power 47
3. Reducing Dynamic Power 49
C. IMPACT OF FAULT TOLERANCE ON POWER USAGE 52

1. Power Cost of TMR 52
2. Alternative Solutions 53
3. Power Advantages of RPR 55

IV. DEVELOPMENT OF A TOTAL PERFORMANCE METRIC 57
A. OBJECTIVE 57
B. CONCEPT 57
C. TOTAL PERFORMANCE METRIC 58
1. Background 58
2. Cost Metrics 59
3. Benefit Metrics 62
4. Total Performance Metric 64
5. Example 1: Generating TPM (Satellite Attitude Control) 65
6. Example 2: Determining K Factors (Satellite Image Processing) 71
7. Accounting for Reliability Factors 75
D. MEASURING RELIABILITY 77
1. Assumptions 79
2. SEU Rates in Space Environment 80
3. SEU Cross Section 83
4. Reliability and Mean Time Between Error 84
5. Reliability Comparison Between RPR and TMR 85
E. SUMMARY 88

V. CORDIC ALGORITHM 91
A. OVERVIEW 91
1. Rationale for Choosing CORDIC 91
2. CORDIC Applications 92
B. MATHEMATICAL FOUNDATION 93
1. Derivation of Circular Rotation Mode 94
2. Selection of Pseudorotation Angles 96
C. ERROR PROPAGATION IN ITERATIVE AND PIPELINED DESIGNS 97
1. Feedback and Error Propagation 98
2. Using TMR and RPR to Correct CORDIC Faults 101

VI. SIMULATION ENVIRONMENT AND RESULTS 105
A. OVERVIEW 105
B. SEU SIMULATIONS 105
1. SEU Simulation Environment 105
2. Test Circuits 108
a. "Davis TMR" Iterative CORDIC 108
b. "Davis RPR" Iterative CORDIC 109
c. "Unprotected" Iterative CORDIC 110
d. "Improved TMR" Iterative CORDIC 111
e. "Improved RPR" Iterative CORDIC 112
3. Results 113
a. Data 113

b. Discussion *118*
C. POWER SIMULATIONS 120
1. Power Simulation Environment 121
2. Test Circuits 124
3. Results 125
D. SUMMARY 129
VII. RADIATION TESTING 131
A. OVERVIEW 131
1. Purpose 131
2. Test Equipment, Setup, and Designs 132
3. Test Procedure 134
B. RESULTS 136
1. Fluence-to-SEU and Cross Section 137
2. Variability of SEUs and Bit Sensitivity 139
3. Polarity of Bit Flips 142
4. Multiple Bit Upsets (MBUs) 145
C. VALIDATION OF SIMULATIONS 146
D. ON-ORBIT RELIABILITY 150
VIII. PRACTICAL RPR IMPLEMENTATION ISSUES 155
A. OVERVIEW 155
B. WORKING WITH UPPER AND LOWER BOUNDS ESTIMATES 155
1. Properties of Numerical Functions Amenable to Approximation 155
2. Lookup Tables versus Direct Calculation 158
3. Improved Method for Generating Lookup Table Estimate Values 163
4. Designing a Voter to Compare Precise and Approximate Calculations 165
C. CONSEQUENCES OF IMPRECISION 167
1. Scenario Description 168
2. Image Compression with Discrete Cosine Transform 169
a. Background *169*
b. Measuring Image Quality *171*
c. MATLAB Testbed *172*
3. Effect of Imprecision on Image Quality 173
4. Affect on Total Performance Metric (TPM) 181
IX. CONCLUSION 187
A. SUMMARY OF RESEARCH 187
B. ORIGINAL CONTRIBUTIONS 189
C. FUTURE WORK 191
D. CONCLUDING REMARKS 194
APPENDIX A – CORDIC PROCESSOR DESIGN 195
A. OVERVIEW 195
B. DESIGN PROCESS 197
1. Schematic Design Specification 197

2. VHDL Design Specification for Iterative CORDIC 199
3. VHDL Design Specification for Pipelined CORDIC 202
C. SYNTHESIS TOOL VARIABILITY 205
APPENDIX B – MATLAB ERROR SIMULATION CODE 209
A. CORDIC ERROR PROPAGATION CODE 209
1. CORDIC Algorithm with Specific Forced Errors 209
2. Supporting MATLAB Code for CORDIC Calculations 211
B. DCT ERROR SIMULATION CODE 214
LIST OF REFERENCES 217
INITIAL DISTRIBUTION 225

LIST OF FIGURES

Figure 2.1 Basic Concurrent Error Detection (CED) Architecture (from [46]) 13
Figure 2.2 Reliability of Serial Systems (left) and Parallel Redundant Systems (right) ... 16
Figure 2.3 Redundancy at System-Level (top) and Component-Level (bottom) 17
Figure 2.4 Complexity-Confidence Relationship (left) and System Structure (right) 18
Figure 2.5 Simple TMR Architecture 21
Figure 2.6 Reliability of NMR Systems 22
Figure 2.7 Simple RPR Architecture 26
Figure 2.8 Bit Significance Distribution Curve 28
Figure 2.9 AND Function Map for 3-Bit, 2-Bit and 1-Bit Input Vectors 32
Figure 2.10 Hypothetical Multi-Function Map 33
Figure 2.11 Clustering for 2-Bit and 3-Bit Representations of Integers 34
Figure 2.12 RPR Suitability Flowchart 35
Figure 3.1 Example of Glitching Behavior 51
Figure 4.1 Hypothetical Cost Curves 60
Figure 4.2 Hypothetical Benefit Curves 62
Figure 4.3 Hypothetical Functional Behavior of TPM Components 69
Figure 4.4 Hypothetical Benefit Curves 73
Figure 4.5 Conceptual Reliability Isocurves within Cost-Benefit Design Space 77
Figure 4.6 SEU Rates for Notional 64 Mbit Memory (from [17]) 81
Figure 4.7 Typical Altitude Variation of SEU Rates for 60° Orbit (from [17]) 83
Figure 4.8 Hypothetical TMR circuit (left) and RPR circuit (right) 86
Figure 5.1 Vector Rotation in 2 Dimensions 94
Figure 5.2 True Rotation (left) versus Pseudorotation (right) 95
Figure 5.3 Iterative CORDIC Hardware Configuration 97
Figure 5.4 General RPR Configuration 101
Figure 6.1 SEU Simulator Configuration with CFTP Hardware 106
Figure 6.2 Layout for “Davis TMR” Circuit 109
Figure 6.3 Layout for “Davis RPR” Circuit 110
Figure 6.4 Layout for “Unprotected” Circuits 111
Figure 6.5 Layout for “Improved TMR” Circuits 112
Figure 6.6 Layout for “Improved RPR” Circuits 113
Figure 6.7 Histogram of Error Counts for Sensitive Bits in “Davis TMR” Circuit 115
Figure 6.8 Detected Sensitive Bit Locations (left) vs. Circuit Layout (right) for “Davis TMR” 117
Figure 6.9 Power Simulation Process Flowchart 123
Figure 6.10 Scatter Plot of Power Consumption vs. Circuit Size 128
Figure 7.1 Hardware Configuration “C1” 133
Figure 7.2 Radiation Beam Test Procedure 135
Figure 7.3 SEU Profile for Run 4 (C1 Hardware) 140
Figure 7.4 SEU Profile for Run 10 (C2 Hardware) 141
Figure 7.5 Locations of Sensitive Bits (Simulator, black) and SEUs (Radiation, red) ... 148
Figure 7.6 Correlation of Sensitive Bit (green) from Simulator and Radiation 149
Figure 8.1 Upper/Lower Bounds for Sine Function 156

Figure 8.2 Upper/Lower Bounds in Vicinity of Stationary Points....158
Figure 8.3 Architecture for Calculating Precise and Upper/Lower Bounds Solutions ...159
Figure 8.4 Approximating Sine Function at Three Sample Points....162
Figure 8.5 Generating Lookup Table Error Bounds for Arbitrary Functions 165
Figure 8.6 Possible Error Conditions in RPR....166
Figure 8.7 Pseudo-code for RPR Voter....167
Figure 8.8 Basis Functions for 8x8 DCT 171
Figure 8.9 "Baboon" Image with Temporary Failure of DCT Processor 174
Figure 8.10 "Gold Hill" Difference Image....175
Figure 8.11 "Gold Hill" Image with Temporary Imprecision in DCT Processor 176
Figure 8.12 Detail of "Gold Hill" Image with Temporary Imprecision in DCT Processor....177
Figure 8.13 "Lena" Image with Persistent Imprecision in DCT Processor....179
Figure 8.14 Detail of "Lena" Image with Persistent Imprecision in DCT Processor 180
Figure 8.15 Benefit Value Functions for Reliability and Precision Factors 183
Figure 8.16 Relationship Between Precision and Reliability TPM Factors....184
Figure A.1 32-Bit CORDIC Processor Schematic....197
Figure A.2 Layout of TMR 32-Bit CORDIC on Virtex XQVR600 199

LIST OF TABLES

Table 2.1 Approximation Methods for 4-Input Sum Calculation....29
Table 2.2 Approximation Methods for 3-Input Product Calculation....29
Table 2.3 Hypothetical Outputs from 3-Modular and 5-Modular Redundancy....41
Table 2.4 Example Relationship Between Alternative Numerical Approximations....42
Table 4.1 Possible Cost/Benefit Prioritization Schemes....69
Table 4.2 Hypothetical TPM for Various Design Points....71
Table 4.3 Example Cost/Benefit Relative Weighting Factors....72
Table 5.1 Pseudorotation Angles for Circular CORDIC Modes....96
Table 5.2 Example Error Propagation in Iterative CORDIC....99
Table 5.3 Example Error Propagation in Pipelined CORDIC....100
Table 5.4 Example Response of Iterative TMR and RPR Designs....102
Table 5.5 Example Response of Pipelined TMR and RPR Designs....103
Table 6.1 SEU Simulator Results for Uncorrected Errors....114
Table 6.2 SEU Simulator Results for Masked Errors....116
Table 6.3 Power Estimates for Unprotected 32-bit Iterative CORDIC Circuit....124
Table 6.4 Power Estimates from XPower....125
Table 6.5 Power Estimates for Modified 16-bit CORDIC Circuits....127
Table 6.6 Dynamic Power Inferred from Power Gradient....127
Table 7.1 Names and Descriptions of Test Circuits....134
Table 7.2 Summary of Radiation Test Results....137
Table 7.3 Fluence-to-Upset by Test Circuit....138
Table 7.4 Comparison of Observed SEU Polarity and Bitstream Values....143
Table 7.5 Example of Manually Verifying SEU Effects....147
Table 7.6 Summary of Radiation Test Results....152
Table 8.1 Initial Attempt at Generating Lookup Tables for Sine Function Approximation....162
Table 8.2 Fixed-Point Two's Complement Number Formats for Example in Figure 8.4....163
Table 8.3 Improved Lookup Table Entries for Sine Function Approximation....163
Table 8.4 Image Metrics for Figure 8.9-Figure 8.13....181
Table A.1 Fixed-Point Two's Complement Number Formats for 32-Bit CORDIC....196
Table A.2 CORDIC Circuit Sizes and Speeds on Virtex XQVR600....206

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ABBREVIATIONS AND ACRONYMS

ASIC – Application Specific Integrated Circuit

CED – Concurrent Error Detection

CFTP – Configurable Fault Tolerant Processor

CMOS – Complementary Metal-Oxide-Semiconductor

CORDIC – COordinate Rotation DIgital Computer

CPLD – Complex Programmable Logic Device

DCT – Discrete Cosine Transform

DSP – Digital Signal Processing

EDAC – Error Detection And Correction

EEPROM – Electrically Erasable Programmable Read-Only Memory

EPROM – Erasable Programmable Read-Only Memory

FPGA – Field-Programmable Gate Array

FSM – Finite State Machine

GAL – Generic Array Logic

LSB – Least Significant Bit

LUT – Look-Up Table

MBU – Multiple Bit Upset

MSB – Most Significant Bit

MTBE – Mean Time Between Error

PAL – Programmable Array Logic

PLA – Programmable Logic Array

PLD – Programmable Logic Device

PROM – Programmable Read-Only Memory

SEE – Single Event Effect

SET – Single Event Transient

SEU – Single Event Upset

SRAM – Static Random Access Memory

TMR – Triple Modular Redundancy

VLSI – Very Large Scale Integration

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

I am eternally grateful for the love, encouragement, and support of my wife, Melissa. An amazing woman, wife, and mother, she made it possible for me to complete this dissertation. As I'm sure she knows, I could not have done this without her. As I hope she knows, my love and thanks for her is everlasting.

Melissa and I are blessed to have three wonderful children, Katie, Ben, and Matt. I thank them for bringing such joy and meaning to our lives. Their energy and enthusiasm towards life is infectious. We are so proud of all their accomplishments and their positive attitude. May they continue to grow and blossom.

I could not have asked for a better dissertation committee, from whom I learned so much. Professors Hersch Loomis, Alan Ross, Jon Butler, Doug Fouts, and Sherif Michael each contributed their unique perspectives and experiences to make my education at NPS extremely rewarding. While pushing for excellence, they all were sincerely committed to helping me succeed. I want to especially thank Professors Loomis and Ross, whose passion for advancing our nation's space program has inspired me and countless others towards ambitious goals.

Many other individuals were instrumental in the success of this research. Through her hard work and persistence, Mindy Surratt from the Naval Research Lab was absolutely essential in making the CFTP program at NPS a reality and enabling my work with actual hardware. I thank her for her amazing dedication, professionalism, and sense of humor as we spent many, many hours staring at computer screens full of ones and zeros. I want to thank my fellow classmate, Tim Meehan, for his advice, encouragement, and assistance. By lending his expertise to the CFTP team, his efforts were also critical to the success of this work. I also want to thank David Rigmaiden and Jim Horning for their efforts in making CFTP a reality.

Finally, I would like to thank my parents, Dave and Julie, for their constant support over the years and for encouraging me in all my endeavors. Without them, I know my life's path would not have led to where I am now.

THIS PAGE INTENTIONALLY LEFT BLANK

EXECUTIVE SUMMARY

Spacecraft computers must operate reliably despite their harsh radiation environment. High-energy protons and other radiation sources cause permanent damage to electrical components as well as transient faults in circuits. Although technologies exist for producing "radiation hardened" devices that are less vulnerable to permanent damage, modern digital electronics are very susceptible to transient errors.

One of the most common manifestations of transient errors is called a single event upset (SEU), or "soft error." This research investigates SEU fault tolerance for a relatively new hardware technology known as field-programmable gate array (FPGA). Due to their unique design, FPGAs require innovative approaches to fault tolerance. Triple modular redundancy (TMR) is commonly used to ensure reliable operation. However, TMR is very costly in terms of chip area and power consumption. This research develops SEU fault-tolerant solutions that use less power than TMR.

This dissertation presents a new fault-tolerant architecture for FPGA circuits called reduced precision redundancy (RPR). RPR is suitable for protecting FPGA circuits on spacecraft against SEUs and achieves improved efficiency by applying redundancy in only the most numerically significant portions of a circuit. This concept, though based on ideas from software engineering, is unique within the FPGA fault-tolerant community. RPR is fundamentally different than TMR because it recognizes that the individual output data bits from a numerical computation often have varying levels of importance. RPR is designed to protect only the most significant data bits, thereby avoiding numerically large data errors. This research shows that the RPR architecture consumes much less chip area and power than TMR designs. In fault-free conditions it provides full-precision results, but when SEUs affect the circuit it produces lower-precision, yet acceptable, output data.

Another unique concept presented in this dissertation is the development of a total performance metric (TPM) for optimizing a system based on numerous performance criteria. The idea behind TPM is that each design parameter (e.g., speed, reliability, size, power, etc.) can be grouped as either a cost or a benefit. These cost and benefit factors

are then related to one another through scaling factors that represent the relative importance of each parameter. This results in a quantitative method for determining an optimal design solution by maximizing the total benefit minus cost.

This research explores the effectiveness of the TMR and RPR architectures by developing and testing circuits on actual FPGA devices. Most of this work is performed with implementations of the Coordinate Rotation Digital Computer (CORDIC) algorithm targeted for the Xilinx Virtex XQVR600, though some tests utilize the Virtex-II XC2V6000 device. The XQVR600 devices tested are part of the NPS Configurable Fault Tolerant Processor (CFTP) space experiment. Results of high-energy proton radiation testing with the CORDIC and other circuits are presented. In addition to proton testing, an automated SEU simulation system is developed, enabling detailed and comprehensive studies of SEU effects without requiring actual radiation sources. This simulator is used to characterize the SEU tolerance of TMR and RPR versions of the CORDIC circuits. Although TMR masks faults affecting any of the output data bits, RPR is designed to ensure accuracy of the most significant bits of the output data. In terms of these two distinct goals, the RPR designs tested here are +/- 30% as effective as the TMR designs.

In addition, power simulations are performed to determine the power cost of implementing TMR and RPR architectures. Though the relative power usage of RPR and TMR depends on many factors, such as the level of precision incorporated into the RPR calculations, the CORDIC circuits tested here show that TMR requires between two and three times as much power. The substantial power savings possible with RPR is justification for accepting some degradation in SEU immunity and/or numerical precision.

Finally, image processing with the discrete cosine transform (DCT) is used as an example application for studying the effects of SEU-induced data errors and imprecision in numerical calculations. Subjective and objective image quality assessments are used to judge the relative importance of preventing errors and maintaining high precision. This image processing example demonstrates that decreased precision in an RPR design causes little performance loss. Thus, it is concluded that RPR is an efficient method for achieving SEU tolerance in FPGA circuits.

I. INTRODUCTION

A. OVERVIEW

Over the past decade the tremendous advances in reconfigurable circuit technologies have spurred considerable interest in using devices such as field-programmable gate arrays (FPGAs) on spacecraft [1], [2], [3]. Although radiation effects are a concern when dealing with any spacecraft electronic system, they are especially troublesome for certain FPGA technologies [1], [4], [5]. In the last several years, an increased awareness of FPGA radiation susceptibilities has spurred development of various mitigation methods, which have proved to be quite effective [5], [6], [7], [8]. However, most of this work has not considered the impact of these methods on a system's power consumption. Power is also an important concern for spacecraft computing and FPGAs generally require more power than application specific integrated circuits (ASICs) to perform comparable functions [9], [10]. Radiation susceptibility and power consumption are impediments to the widescale use of FPGAs for spacecraft [3], [9], [10]. This dissertation investigates a new technique for mitigating radiation effects while minimizing the impact on a spacecraft's power budget. The methods proposed in this research offer substantial benefits for various computing problems for both FPGA and non-FPGA systems.

B. BACKGROUND

1. Computing In the Space Environment

Since the launch of Sputnik I/II in 1957 and Explorer I in 1958, satellites have grown tremendously in importance for both military and civil uses. Satellites are widely used in communications, navigation, weather monitoring, earth remote sensing, missile warning, and many other applications. Computers are an integral part of all spacecraft. Although the earliest spacecraft were simpler and much less capable than today's spacecraft, even the first satellites had rudimentary computing systems. By 1962 NASA was developing rather sophisticated computer systems for their space projects [11]. Virtually all satellites require some on-board processing capability, whether for complex data processing or simple satellite control functions.

The demand for satellites with increasingly complex functionality creates a need for greater on-board computer processing capacity [12]. For many spacecraft, the amount of data collected by on-board sensors greatly exceeds the available downlink bandwidth [13]. In data-intensive applications, such as synthetic aperture and imaging radars, the ability to process large amounts of raw data can significantly reduce the downlink requirements [14], [15]. A prime example demonstrating the benefits of on-board processing is Los Alamos National Lab's current Cibola spacecraft program [16]. A primary mission of the Cibola spacecraft is to analyze lightning events using a reconfigurable radio operating at VHF and UHF frequencies. This system requires 2.4 Gbps processing capacity, which is achieved using 9 Virtex XQVR1000 FPGAs operating in parallel on the spacecraft [2]. If all data processing were performed on the ground, it would be extremely difficult to support downlink of the raw data at such high speeds. In fact, Cibola's downlink capabilities are limited to 38.4 kbps and 4 Mbps links, far less than the 2.4 Gbps raw datastream. Furthermore, on-board processing enables persistent global coverage since the finished data products will likely be small enough to store on-board in between direct line-of-sight communication periods, whereas the raw data is far too voluminous for on-board storage. In this and many other modern space-based missions, spacecraft processing is absolutely vital.

The tremendous cost of launching systems into space imposes stringent design constraints on the size, weight, and power of satellites. Satellites must operate remotely for long periods of time. High reliability is essential, as most are not serviceable after launch (the Hubble Space Telescope is the most noteworthy exception). These challenges, common to all spacecraft, contribute to long development schedules and high costs. Due to the large investments associated with spacecraft programs, there is tremendous pressure to make the systems highly reliable. This need for reliability lengthens the procurement process and further drives up the costs, perpetuating the cycle. In addition, these factors create an obsolescence problem. By the time a satellite is ready to launch, technology has advanced far beyond what is integrated into the satellite. By using out-dated technology, satellite programs also suffer from a lack of commercial support for obsolete equipment and spare parts.

The space environment also presents considerable challenges to spacecraft electronics. Modern electronic systems, and computers in particular, are sensitive to the high radiation levels in space. In the early days of spacecraft computing, there was little concern about radiation effects. For example, a 1966 paper acknowledged the potential hazards of the space environment but concluded that, "data have shown no errors that could be attributed to interference from outside the spacecraft." [13] However, as solid state electronics rapidly advanced in capabilities and shrank in physical dimensions, radiation effects became an issue for computer systems. By 1975 there was some limited evidence of radiation-induced computer faults on spacecraft. By 1978 such effects were even observed in some terrestrial systems and the radiation effects community began in earnest to explore this phenomenology [17].

Radiation sources, such as highly energetic particles trapped in the earth's magnetosphere, can cause permanent damage to electrical components as well as transient faults in circuits. Permanent damage to electronic devices can result from various physical phenomena, but is primarily caused by accumulation of trapped charges from ionizing radiation and atomic displacement from non-ionizing radiation [17]. Temporary effects are primarily caused by energetic particles that locally generate a large number of excess charges in a device and thereby affect signal values in a circuit [17], [18]. Spacecraft computer systems must operate reliably in spite of these challenges.

2. Single Event Upset (SEU)

Spacecraft electronics are exposed to substantially more damaging radiation than terrestrial systems, which are protected in large part by the earth's atmosphere and magnetosphere [17]. This radiation can lead to both permanent and temporary effects. While protecting electronics against permanent damage is very important, this research focuses on temporary effects. The majority of temporary, non-persistent perturbations in spacecraft electronics occur nearly instantaneously and are caused by individual high-energy particles that affect a localized region in the semiconductor device. The term "single-event-transient" describes these rapid and isolated events.

Single-event-transient (SET) effects, such as the well-known single event upset (SEU), can be caused by any undesirable energy source, such as corpuscular radiation or

electromagnetic interference. Clark [19] presents succinct definitions of SET and SEU: "SET…is an unintended analog pulse" and "SEU occurs when an SET causes a bit-flip error in a memory element." SET describes a physical process that occurs in an electrical circuit. Depending on many factors, such as circuit design and precise timing of SET events, this process may or may not lead to an SEU. Similarly, an SEU may or may not lead to erroneous data at the system output [19]. This research is primarily concerned not with the physical causes and effects of SETs, but rather with the consequences of SEUs on logic circuits.

Transient faults are also commonly called Single Event Effects (SEEs) [3]. Within the category of SEE, some researchers differentiate between a Single Event Functional Interrupt (SEFI), which causes the device to "lock-up" and cease functioning until a complete reset or power-cycling is performed [3], [5], and an SEU, which causes a bit-flip. For simplicity and consistency with most of the literature, the term SEU henceforth refers to all transient-induced logic faults.

SEUs are a serious concern for space systems. Exposure to high radiation levels leads to SEUs in spacecraft computers that can cause erroneous results. Without mitigation of SEUs, the consequences of such errors can range from relatively benign data errors to catastrophic effects such as loss of satellite control and functionality [3], [20]. Considerable effort is spent to mitigate SEUs in modern satellite systems through techniques such as shielding, error detecting and correcting codes, and hardware redundancy.

Concern over SEUs and their impact is not limited to spacecraft systems. In fact, there is growing concern that terrestrial computers are becoming more prone to radiation-induced faults. Demand for high-performance processing capabilities drives requirements for smaller and more sophisticated computers. Many of the factors that improve computing performance also make state-of-the-art systems increasingly sensitive to even low levels of radiation [17], [21]. Alpha particle radiation from naturally occurring radioactivity in packaging material has long been known to cause SEUs, also known as "soft errors" [17]. Shrinking integrated circuit feature sizes, operating voltages and noise margins exacerbate the SEU problem. Thus, logic upsets caused by alpha

particles, cosmic rays and other radiation sources are becoming more of a concern for ground-based commercial systems [21], [22].

Therefore, technologies are needed to ensure that both spaceborne and terrestrial computers can be highly reliable and still meet their speed, size, weight, power, cost and other design constraints. Techniques that improve the SEU immunity of electronics are of significant value in military, civilian, and commercial applications. Increasingly, SEU tolerance is becoming a common feature for many digital systems.

3. Field-Programmable Gate Array (FPGA)

Reconfigurable circuit technology is revolutionizing the computing industry. Traditionally, hardware consisted of fixed circuit configurations and software provided much of a system's design flexibility. Modern configurable logic structures allow hardware circuits to be reconfigured post-production and even during operation. Programmable logic devices (PLDs) first appeared in the 1970s with the introduction of the PAL and quickly evolved into numerous products with names such as PLA, GAL, PROM, EPROM, and EEPROM [23]. However, these earlier technologies had limited utility because they offered relatively small gate counts, had fixed interconnect structures, and were typically only one-time programmable. The advent of FPGAs in 1985 [24] established a configurable circuit technology that could perform even complex functions and maintain great flexibility. The most distinguishing traits of FPGAs compared to other technologies are their relatively large numbers of logic gates and very flexible interconnect architecture. FPGAs have benefited from rapid advances in VLSI technologies and now rival ASIC and general purpose microprocessors in numerous applications [25].

The tremendous advantages of reconfigurable circuits have prompted great interest from the space computing community. The first published description of an FPGA used in space concerned the SAMPEX spacecraft, launched in 1992 [26]. Although the earliest FPGA devices were built for one-time programmability, there is now more demand for reprogrammable devices, such as those using Static Random Access Memory (SRAM) cells for storing the circuit configuration. Other technologies are also widely used to construct FPGAs. For example, anti-fuse devices are configured

by applying high-voltage to selected nodes in order to change them from their initial high-impedance state to a low-impedance "fused" state, in much the same way as PROMs are programmed. Spacecraft have used anti-fuse FPGAs for many years and have recently begun to utilize SRAM-based devices [3]. This research focuses on the SRAM devices and does not address those using anti-fuse and similar one-time programmable technologies. Thus, throughout this dissertation the term FPGA implies SRAM-based devices only. Over time, these devices will likely gain broader acceptance and become standard components for on-board computing.

The flexibility of SRAM-based FPGAs offers numerous advantages. First, a single FPGA device can perform multiple hardware functions that would otherwise require multiple devices. By loading different circuit designs onto the chip when needed, one can achieve broad functionality with fewer parts than conventional systems. In SRAM-based devices this switching back and forth among configurations can be accomplished very quickly, on the order of a millisecond. Another key advantage is the ability to correct design deficiencies, compensate for permanent hardware failures, and make enhancements long after system deployment. Software and hardware are prone to design errors or oversights ("bugs"). Design errors are sometimes discovered long after initial production. For example, NASA's Cassini-Huygens mission to Saturn contained a critical communication design flaw that wasn't discovered until the probe was over 430,000,000 km from earth [27]. An oversight related to the Doppler-shifted datastream coming from the Huygens probe during descent towards the moon Titan would have caused the Cassini spacecraft to receive only a scrambled version of the data. While minor changes to on-board firmware could have fixed this problem, the only option after launch was to redesign the spacecrafts' flight paths, thereby losing some of the mission's scientific value, to minimize the Doppler effect. With FPGAs, the hardware can easily be reconfigured even after launch to correct or compensate for any such problems.

Unfortunately, reconfigurable SRAM-based FPGAs are susceptible to SEUs. SEUs can cause data bit upsets, as in other circuit technologies, but can also affect circuit operation since the circuit configuration is stored in volatile memory. Thus, fault-tolerant schemes for FPGA circuits must compensate for both data and configuration memory

upsets. Error detection is an important aspect of the fault-tolerant designs explored in this research, as it enables the correction of both types of upsets.

When FPGAs, or the systems in which they are used, experience permanent physical damage, the FPGAs can be reprogrammed to avoid the faulty components [28]. For example, faults within the FPGA can be circumvented by rerouting the circuit. Furthermore, the I/O pins or the circuit behavior within the FPGA can be modified to compensate for problems outside the FPGA. This reprogrammability also allows for future improvements to the algorithms and design. The popularity of spiral development (the process by which a system is fully developed over several product generations with ever-increasing capabilities) in high-tech programs indicates that this feature of FPGA-based systems has tremendous value. In addition, FPGAs can accelerate development time compared to the traditional approach of building custom ASIC chips for complex electronic systems. FPGAs can be readily purchased on the commercial market but can be fully customized when assembled into the full system.

4. Power Consumption

Power-efficiency in electronic systems is becoming increasingly important as computers find more applications in small and power-limited systems, such as cell phones and smart cards [29]. Power consumption also presents thermal issues related to heat dissipation in devices both very small (like laptop computers) and very large (such as mainframe supercomputers) [30]. Especially in the rapidly growing portable electronics industry, there is considerable interest in developing energy efficient designs [31]. One of the reasons that FPGAs have not supplanted ASICs more quickly is that they typically use significantly more power [10].

Power generation and energy storage are significant limitations in remote systems like spacecraft. Virtually all current satellites use solar cells for power generation. These power sources are limited by the collection area available for body or panel-mounted solar cells. Even on large spacecraft such as the International Space Station with hundreds of square meters of solar panels, power is limited to 10’s of kilowatts. The most powerful commercial satellites produce roughly 20 kW, but small satellites such as

NPSAT have power budgets of only 10s of watts that must be shared among the various subsystems. Spacecraft electronics must be specifically designed for these extremely low-power conditions.

Spacecraft typically use batteries for energy storage, though momentum wheels and other technologies have been investigated. Issues with energy storage methods include total capacity, conversion efficiency, peak discharge rate and longevity. For example, the battery system on PANSAT, the first satellite built at NPS, was the first major subsystem to fail, thereby cutting short the lifetime of this otherwise healthy satellite [32].

Thermal dissipation is also a major concern because common cooling techniques, such as convection with fan-driven airflow, do not work in the vacuum of space. In addition, the temperature cycles that many spacecraft undergo due to fluctuating solar illumination further complicate the thermal balance inside spacecraft. Removing heat from high-temperature computer chips on satellites often requires considerable effort. For example, careful thermal design was needed on the Cibola spacecraft to provide heat dissipation for several FPGAs consuming over 7 watts each [33].

C. MOTIVATION

Spacecraft computers face the unique challenge of operating with a limited power budget and functioning reliably in a high radiation environment. Numerous studies have examined trade-offs between size vs. speed, fault tolerance vs. size, power vs. speed, power vs. size, etc., for specific computation problems [34], [35], [36], [37], [38]. Relatively little research has been done to examine power efficiency in fault-tolerant architectures. Traditional approaches to fault tolerance assume that reliability is paramount and power consumption is a secondary concern [21]. However, Maheshwari [29] identifies both fault tolerance and low-power as “key objectives in the design of critical embedded systems.” Another research team [4] suggests that in some applications it may be beneficial to trade off reliability for power. This research explores these design trade-offs and presents a new approach for achieving high system reliability using minimal power.

Several key observations inspired this research project. First, SRAM-based FPGAs, which offer great advantages for space computing, have unique failure mechanisms that render many common fault-mitigation techniques ineffective [4]. Power is a major limitation in spacecraft computing, making reduced power consumption an important design goal. Triple Module Redundancy (TMR), the most common fault-tolerant technique for FPGAs, is very costly and is wasteful in many applications. Finally, many computational tasks can accept "flexible precision" without significant degradation. This dissertation proposes a unique computing architecture called Reduced Precision Redundancy that is suitable for FPGAs and provides sufficient fault tolerance with minimal power consumption.

D. CONTRIBUTIONS

The primary contribution of this dissertation is the development of a new fault-tolerant architecture for FPGAs called Reduced Precision Redundancy (RPR). This new architecture can be applied to a variety of computational tasks and minimizes the cost of hardware fault tolerance. RPR is unique because it applies software-based fault tolerance concepts in a hardware fault-tolerant structure. While other methods exist for building hardware fault tolerance with reduced overhead costs, RPR is unique because it prevents large data errors from propagating through the system. Other low-cost approaches permit large and small data errors with equal likelihood.

Second, this dissertation demonstrates that RPR generally provides better overall performance than TMR, considering both benefits (performance, reliability, etc.) and costs (chip area, power, etc.). This research develops unique methods for determining the optimal balance between fault tolerance and resource usage.

Third, this research produces a validated process for assessing SEU fault tolerance of RPR and other designs implemented on the Configurable Fault Tolerant Processor (CFTP) experiment's Xilinx XQVR600 devices. CFTP is an active research project at the Naval Postgraduate School (NPS) that is building two space experiments as part of a program investigating reliable and reconfigurable computing. Part of this dissertation involved expanding the capabilities of an SEU simulation system developed at the Naval

Postgraduate School, enabling rapid and complete characterization of specific circuit designs. Live testing at a radiation test facility validated the simulation system. Finally, analytical methods were developed for assessing the severity of various error conditions in an RPR architecture.

E. DISSERTATION ORGANIZATION

Chapter II presents background material on fault tolerance for FPGAs and introduces the RPR architecture. The chapter describes the characteristics of algorithms and applications that are well suited for use with RPR. **Chapter III** describes approaches for reducing electrical power consumption in FPGA designs. **Chapter IV** develops a quantitative method for determining how to best achieve a low-power and fault tolerant system. While the focus of this research is on SEU fault tolerance and power consumption, the methodology is flexible and can be expanded to address a multitude of competing design goals. **Chapter V** briefly describes the CORDIC algorithm and why it was used as the primary demonstration system in this research. **Chapter VI** describes the SEU and power simulation environments and provides accurate predictions of RPR performance in radiation environments. **Chapter VII** describes the proton radiation testing performed to validate the SEU simulation environment of Chapter VI. **Chapter VIII** discusses some practical issues that must be addressed when creating an RPR circuit. The chapter also examines an image compression problem as a case study for demonstrating the viability of RPR. **Chapter IX** summarizes the dissertation and identifies areas for further investigation.

II. FAULT-TOLERANT DESIGN CONCEPTS

A. FAULT TOLERANCE FOR FPGAS

Parhami defines fault-tolerant computing as "ensuring correct functioning of digital systems in the presence of (permanent and transient) faults" [30]. For the purposes of this dissertation, it is important to distinguish between faults and errors. Faults are the undesired changes that occur in a physical circuit/device that may lead to incorrect operation. Errors occur when a circuit produces incorrect output data [39]. Faults, though undesirable, are not necessarily harmful. Our intent is to prevent errors by properly managing faults.

This research focuses on developing fault-tolerant techniques for FPGAs. Although computer fault tolerance in various forms has been investigated for over six decades [40], FPGAs have several unique features that necessitate new fault-tolerant methods for these increasingly important devices [41]. FPGAs used in spacecraft face additional challenges unique to the hostile operating environment of space.

Although FPGAs have only become widely used in the last decade, there has been substantial research and development in fault-tolerant techniques for these devices. While it is important to consider all potential sources of faults when designing high-reliability systems, this research focuses specifically on faults from SEUs and does not consider other failure mechanisms such as device burnout and power supply glitches.

In SRAM FPGA technology, both the data being processed and the circuit function itself are stored in memory elements. An SEU can flip a data bit, corrupting a portion of the data stream, or it can flip a configuration bit, causing the circuit to no longer behave in the intended manner [42]. Errors in an FPGA's configuration memory can modify the circuit functionality and produce incorrect results, even if the stored data bits are correct. Errors affecting circuit function, which are extremely disruptive, occur more frequently in modern FPGAs than data errors. One study of the Virtex chips concluded that 91% of static upsets (i.e., those observed without a clocking signal) were attributable to the "configuration" bits stored on the device [43]. The authors in [3] note that user data latches comprise a relatively small percentage of SRAM latches in the

Xilinx 4000 chips and point to a test that found roughly 80 configuration upsets for each user data upset. Configuration errors are the primary concern for this research, although protection against data errors is an important secondary objective.

Conventional fault mitigation techniques are insufficient for dealing with the broad range of possible faults in FPGAs [4]. Fault models such as "single signal stuck-at-0/1" and "stuck-open" do not adequately address the wide variety of possible malfunctions in FPGA circuits [44], [45]. Error detection and correction (EDAC) codes such as Hamming codes are effective for detecting and correcting small numbers of data errors in storage or transmission and to a limited extent in processing. Such codes are extensively used in digital memory chips and communication systems. However, configuration errors in FPGAs can easily exceed the capability of EDAC techniques. FPGA designs require unique solutions that provide fault tolerance against a wide range of potential data and configuration fault conditions.

B. PRINCIPLES OF FAULT TOLERANCE

1. Fault/Error Detection vs. Correction

The first step towards fault-tolerant systems is fault and/or error detection. Without some means of determining that a fault exists in a circuit, there is no way of fixing that fault or performing error correction. In some situations it may be acceptable to simply identify the presence of faults or errors. For example, if only a few data samples in a sensor datastream are corrupted, one solution may be to simply flag errors and then discard the bad data. In other situations, the data is too critical to be discarded and must be corrected, either in real-time or in post-processing. Error correction is more difficult, but fundamentally relies on fault/error detection.

2. Concurrent Error Detection and Checkpointing

Concurrent error detection (CED) involves the discovery of faults/errors as part of the data computation process. The goal with CED is for the circuit to flag errors before incorrect data is propagated through the system, thus maintaining "data integrity" [46]. CED "is designed to detect the first error produced by a failure in the system and is therefore capable of detecting permanent and transient faults." [47] For hardware fault

tolerance, spatial redundancy with result checking is typically used, as shown in Figure 2.1. A common technique is a duplex system in which two modules compute the same function and the results are compared for equality. A mismatch in the comparator signals that one of the modules (or the comparator) is faulty; thus providing fault detection but not error correction. Temporal redundancy can also be used, however this adds considerable latency and often reduces system performance. Most common fault-tolerant methods, such as TMR, are variants of the basic CED structure. CED is an implicit feature of the fault-tolerant designs explored later in this dissertation.

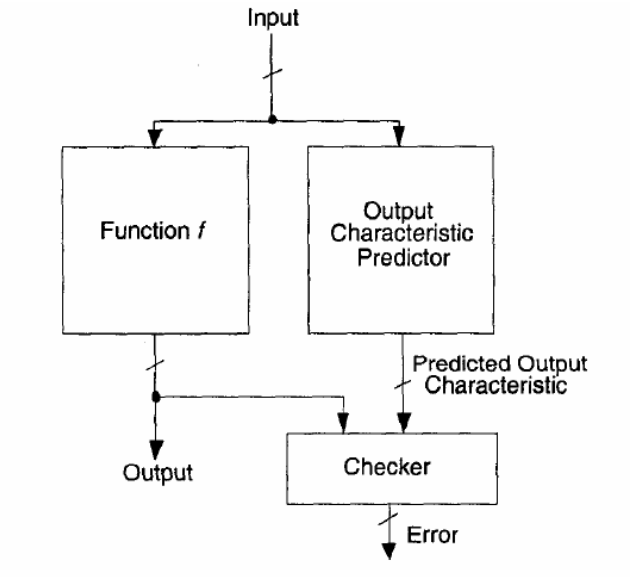


Figure 2.1 Basic Concurrent Error Detection (CED) Architecture (from [46])

Another technique that is common in software engineering, called "checkpointing" (or "roll-back and recompute"), applies an acceptance test verifying that each output is valid before passing on the result. This acceptance test is based on some *a priori* knowledge of the function being calculated or properties of allowable outputs. If an output is invalid, the processor "rolls back" and repeats the calculation using the original input. This is in contrast to CED in which the process "rolls forward" when faults are detected instead of recomputing the original inputs.

Checkpointing is popular in software designs, in part because it only requires a single processing element (usually the microprocessor) and enough memory to store the

system's state between checkpoints. However, checkpointing alone is not capable of correcting SEU faults in FPGAs. As discussed earlier, the majority of SEU-induced faults affect configuration memory. Configuration faults persist until the device is reconfigured to correct these faults. Simply repeating the same calculation, as in conventional checkpointing, would only continue to produce erroneous results.

Both CED and checkpointing methods lack fault correction, a crucial element for SEU fault tolerance in FPGAs. Because FPGA configuration faults are both common and persistent, a means of removing such faults is essential [42]. The process of finding and fixing these faults is often called "configuration scrubbing," or simply "scrubbing."

3. Configuration Scrubbing

Configuration scrubbing can be described as "the transparent process of reloading the configuration bitstream so upsets are corrected." [48] In its simplest form, scrubbing involves periodic reconfiguration of the entire device to restore the chip to the desired configuration, regardless of whether or not any SEU faults exist [42], [49]. More sophisticated methods involve refreshing the configuration memory contents only when cued by some fault/error detector. For example, the NPS Configurable Fault Tolerant Processor (CFTP) approach involves periodic *reading* of the configuration memory and only *reloading* the content when sensitive bit upsets (i.e., error producing configuration faults) or a certain number of non-sensitive upsets are detected. In this context, the term configuration scrubbing includes both the reading and reloading functions.

In conventional systems, this scrubbing process involves downtime while the device is reconfigured. While this can degrade overall performance, modern Xilinx Virtex devices support a mode called "active reconfiguration" that permits the reading and writing of configuration bits concurrently with regular device operation. The frequency of such a process might be determined based on an expected upset rate (number of upsets per second) or simply based on the fastest possible reconfiguration speed for the device (bits per second). Faster reconfiguration is desirable since it minimizes the time during which output errors might persist, but continuous configuration scrubbing will consume additional power. The minimum scrubbing cycle duration is determined by the speed of reading/writing configuration bits and the size of

the circuit to be scrubbed. More sophisticated scrubbing techniques involve fault isolation to determine which specific portions of the chip require repair. By pinpointing faults, the scrubbing process can be sped up considerably while minimizing power consumption.

4. Redundancy

Redundancy is essential for fault tolerance. Without redundancy, a system's reliability is limited by the product of the individual reliabilities of each subcomponent, as shown in Equation 2.1 [50]. In this equation R_i represents the probability that subcomponent i will function correctly at any given time. This equation applies to systems in which components are connected in a serial fashion. Like the links in a chain, failure of a single component can cause the entire system to fail.

$$R(t) = \prod_{i=1}^{N} R_i(t) \tag{2.1}$$

When a system's components are replicated and properly integrated in a parallel style, a failure in one component can be "masked" by redundant elements that continue to function properly. This parallel structure can dramatically improve reliability as shown in Equation 2.2 [50].

$$R(t) = 1 - \prod_{i=1}^{N} \left[1 - R_i(t)\right] \tag{2.2}$$

Figure 2.2 illustrates how these reliability functions depend strongly on the values of N and R_i. The graph on the left shows how the overall reliability of a system of several components configured in a serial manner degrades rapidly as N increases. Each curve represents a different individual component reliability, ranging from 0.91 to 0.99. In order to construct highly reliable complex systems with many subcomponents, each component must be very reliable. The graph on the right shows how a system with 10 components (using the rightmost data points on the left graph as the baseline reliability) can be made more reliable through parallel redundancy. Note that for the upper curve with double or greater redundancy, the "system of systems" is more reliable (>0.991) than any single component by itself (0.99).

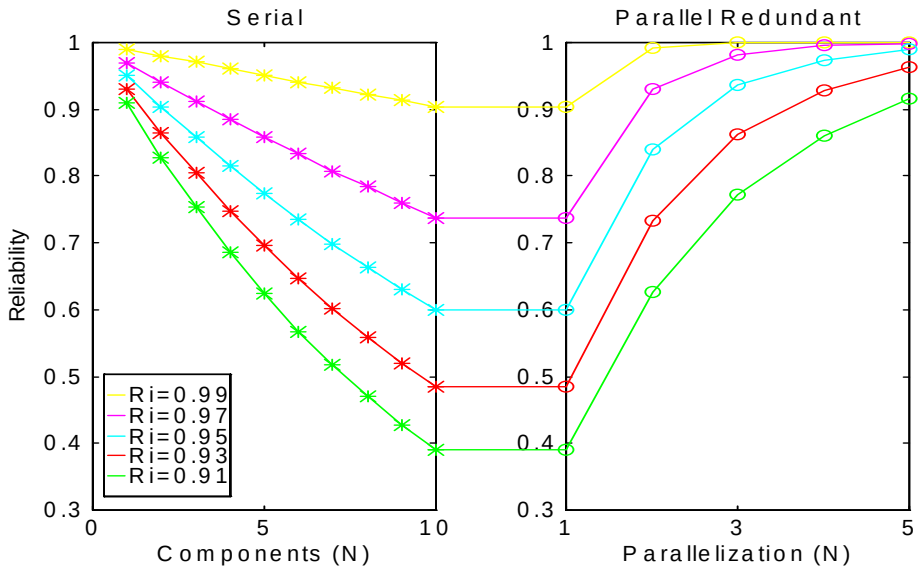


Figure 2.2 Reliability of Serial Systems (left) and Parallel Redundant Systems (right)

Reliability of computer systems can be improved through spatial redundancy, temporal redundancy, or a combination of the two. Spatial redundancy is the most common means of enhancing hardware reliability. It can be applied at a top level by replicating the entire system or at lower levels by replicating subcomponents. EDAC techniques are a type of spatial redundancy that primarily resolve data errors. Triple modular redundancy is a common method used for masking both logic and memory faults. Temporal redundancy consists of multiple calculations that are performed in a time sequence before a final result is determined. The checkpointing technique described earlier is a form of temporal redundancy.

a. Configuring Redundant Components

Another important observation is that redundancy is most effective when applied at the lowest possible level in a system. The following figure shows two possible arrangements for a double redundant system with 5 serial elements. The first configuration uses redundancy at the highest level with the entire system duplicated. The second configuration applies redundancy to each element in the overall structure.

Assuming each component has a reliability of 0.95, the first configuration yields an overall reliability of 0.949 while the second configuration is much better with reliability of 0.988.

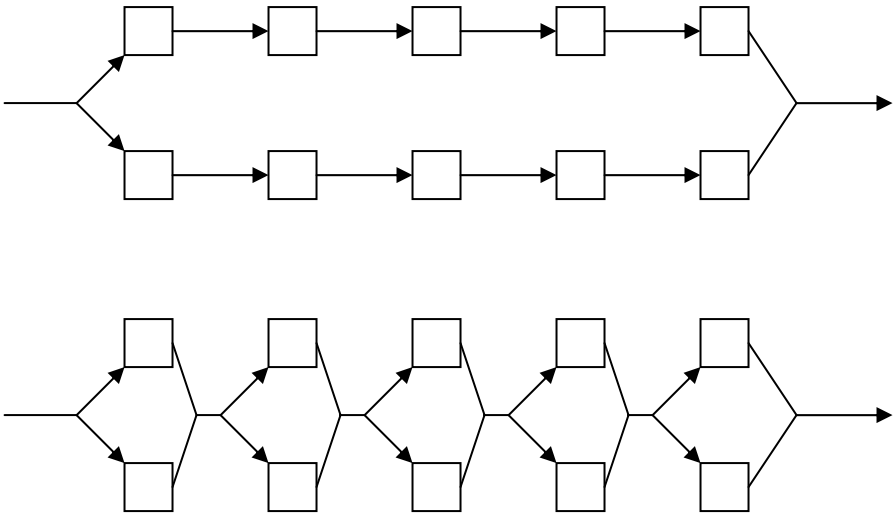


Figure 2.3 Redundancy at System-Level (top) and Component-Level (bottom)

b. Selective Redundancy

Another important consideration in designing fault-tolerant systems is that redundancy should be applied where it can provide maximum gain for minimum cost. In other words, a fault-tolerant design should be efficient. While some form of redundancy is necessary to protect against SEUs, not all bits of information in a circuit have the same degree of importance. Efficiency is achieved by applying redundancy to only the most important portions of the design.

Though the idea of selective redundancy has existed for decades [39], it has recently gained more attention from the FPGA fault tolerance community. Several research projects [51], [52], [53] have looked at ways of identifying the most sensitive portions of FPGA circuit designs. These sensitive elements, such as critical finite-state machines and feedback structures, can then be targeted for TMR protection using less overall redundancy. While selective TMR can reduce overhead costs, it is typically less effective than full TMR. Thus, there is a large trade-space for optimizing fault tolerance and overhead costs.

5. Complexity/Confidence Trade-offs

The idea that complexity and confidence are opposing parameters is important in developing more efficient fault-tolerant designs. This concept, shown on the left side of Figure 2.4, expresses the intuitive idea that as the complexity of a problem increases, so does the probability of making an error. Therefore, one has less confidence in the accuracy of the result. In computing systems, higher precision solutions require greater complexity. The boxes in the graph represent component count, which was shown in the previous section to adversely affect reliability. To produce highly precise information, a system must perform more calculations and handle more data. This complex processing creates more opportunities for error – *higher precision is associated with lower confidence*. Conversely, the data from simpler calculations is less precise but more reliable. As one moves to the left on this curve, output data becomes more reliable but loses precision. The axis labeled "Complexity" can also represent precision, information, effort, size or power. A simple circuit using only a few components requires relatively little chip area and power. As the circuit is made more complex in order to generate more detailed results, it will require more area and power. In addition, each new circuit element introduces new opportunities for faults and errors to occur.

The pyramid shape on the right side of the figure shows one way of thinking about how a system is constructed. The bottom of the pyramid contains many components, making these lower levels more complex but less reliable. The top of the pyramid is more reliable, but with fewer components and less complexity these top levels cannot provide high precision results.

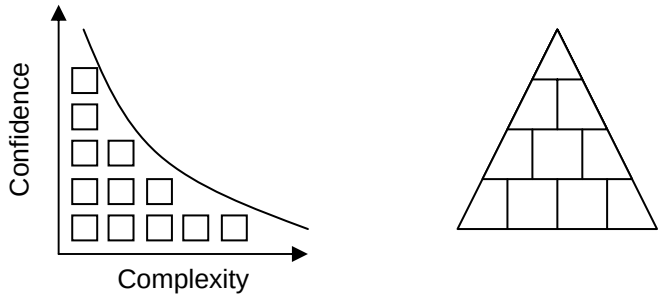


Figure 2.4 Complexity-Confidence Relationship (left) and System Structure (right)

Consider, for example, the calculation of the trigonometric functions sine and cosine. Given any angle, one can be certain that both the sine and cosine of that angle are in the range [–1, +1]. This is not very detailed information, but it comes with high confidence. A simple hand calculation could improve the estimate of the sine and cosine values to perhaps within +/- 0.1 or better. While the second estimate is more precise, its solution requires more calculations, making the estimate more prone to errors. Therefore, we have less confidence in the more precise value. For certain algorithms, it may be sufficient to know that the sine or cosine value falls within the range [–1, +1] or that the function is positive or negative. An "exact" value (for example, precise to 10 decimal places) may be desirable but not essential. In many cases being within an approximate range is more important than having a highly precise, but possibly incorrect, result.

In order to be both reliable and precise, fault-tolerant computer designs must include subcomponents that span the confidence-complexity spectrum. The main processing blocks form the lower levels in the pyramid and perform the detailed calculations that produce high precision results. However, a typical processing block provides no protection against functional faults and is therefore relatively unreliable. Voting circuits resolve potential disagreement among the redundant processing elements and determine which result to accept. Control circuitry manages the flow of information through the processing blocks. These voter and control circuits represent the top of the pyramid. They contribute little information toward the intended system function, but must be extremely trustworthy.

While neither the voter nor the control circuits produce useful output data, a failure in either of these units can invalidate the entire fault-tolerant structure. These system elements can be thought of as the "reliable kernel" or "trusted agent." Because the reliable kernel performs vital functions that are essential to system operation, this kernel must somehow be made more trustworthy than other parts of the system. In redundant computing systems, the reliable kernel needs to include the voter circuitry and as many of the "single points of failure" as possible.

C. ERROR CODING TECHNIQUES

An interesting option for FPGA fault tolerance is to employ error coding techniques. Berger and residue codes for fault tolerance have long been considered more efficient alternatives to TMR, which is discussed in the next section. However, their limitations and overhead requirements make them unattractive from a cost perspective for controlling faults in complex FPGA designs.

Berger codes are not suitable for the type of fault tolerance addressed in this dissertation. First, they only provide error detection; they cannot locate or correct errors. The Berger checks are designed to trigger an error flag whenever numerical errors occur, but a higher-level system is needed to correct the errors. Second, Berger codes rely on the assumption that all errors in the data word are unidirectional, that is either all 0-to-1 transitions or vice versa [46]. Such a fault model is insufficient for the complex interactions within FPGA circuits. Third, Berger codes are more costly than simple duplication [46], [47]. Their area overhead is often nearly the same as the duplication method, though Rao mentions that overhead can be lowered if the Berger coded data can be directly relayed to other modules without being decoded. In various studies, Berger coding introduced between 50% and 250% area overhead. Finally, Berger coding requires significant modification of the original circuit, whereas duplication techniques such as TMR are much simpler to implement.

In [54] a biresidue code technique is proposed for conserving circuit cost, size and power relative to the common triplicated redundancy method that Rao describes as the "von Neumann approach." Rao estimates that the additional circuitry needed for biresidue checking is roughly equal to duplication of the original arithmetic circuit. Though better than TMR, this amount of redundancy is still quite significant. The complexity and reliance on encoding, decoding and checking units is an even greater cause for concern. In practical applications, the moduli are typically of the form 2^k or $2^k \pm 1$ to simplify the calculations. Nonetheless, calculating residues can be computationally expensive. Furthermore, it is difficult to ensure correct operation of the residue checkers in an FPGA system without making them redundant as well. Correcting detected errors involves recomputing missing data [54], which may require circuits nearly as complex as those being protected.

Another issue with residue check approaches is that they don't often provide graceful degradation. The biresidue approach in [54] will detect and completely correct a wide range of possible faults, but the errors whose residue is zero go undetected. Since a residue of zero only occurs for numbers that are integer multiples of the product of the base moduli, these undetected errors differ greatly from the correct solution. Thus, this approach does not make use of the varying importance of the different bits in an output data word.

D. TRIPLE MODULAR REDUNDANCY (TMR)

A common technique for improving reliability within FPGAs is known as Triple Modular Redundancy (TMR). The most straight-forward TMR style is to replicate the desired circuit three times and include voting logic to determine the most likely correct output. Because the most likely fault scenario is that a single module is in error, agreement between two modules is enough to determine the correct result. Thus the voter outputs the most common result. Figure 2.5 shows a simple TMR architecture. Note that the voter module should be considered part of the "reliable kernel" since a failure here can directly corrupt the output data.

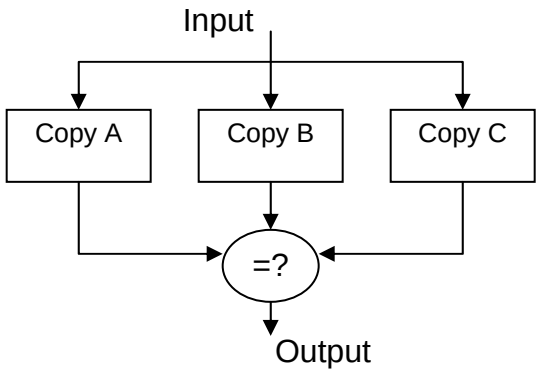


Figure 2.5 Simple TMR Architecture

In general, it is possible to build N-modular redundant (NMR) systems with increasing reliability as N gets larger. Figure 2.6 shows this improvement as N increases.

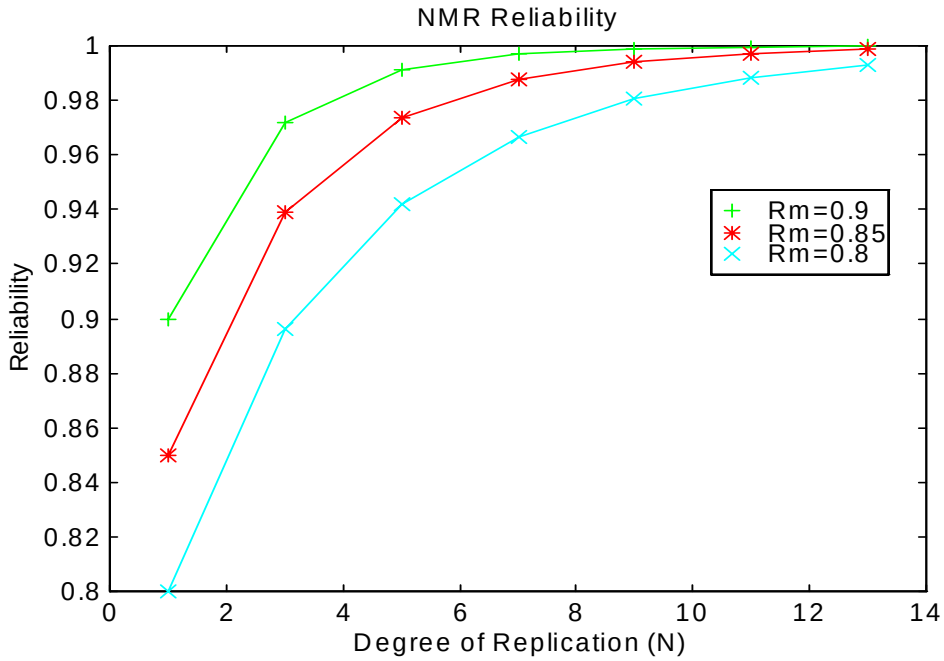


Figure 2.6 Reliability of NMR Systems

However, the reliability improvement is not a linear function of N. The relative gain decreases as N increases, as seen in the figure. Equation 2.3 describes the performance of an NMR system in which each module has the same individual reliability, R_m [50]. In this equation M represents the number of modules out of the total number N that must simultaneously operate correctly for the system to work; typically M is greater than ½ of N. A realistic reliability model must also include the possibility of voter and other shared logic failure. These factors can be included as multiplicative terms that reduce the reliability estimate [39].

$$R_{M\,.of\,.N}(t) = \sum_{i=0}^{N-M} \left(\frac{N!}{(N-i)!i!} \right) R_m^{N-i}(t)\left[1 - R_m(t)\right]^i \qquad (2.3)$$

By triplicating the logic and adding the voter/control circuitry, a TMR fault-tolerant design uses more than 3 times the chip area and power of the original circuit. Rollins, et al., tested several circuits and found that the TMR circuits used 3x-7x as much power as the original circuits [9]. A TMR design is wasteful in that it performs each

calculation 3 times but only uses each calculation once. This can be described as a "high redundancy factor." Conversely, a design with a "low redundancy factor" and an effective fault mitigation technique could provide high reliability with lower overhead costs. Therefore TMR is not always the best choice for meeting overall system requirements.

E. REDUCED PRECISION REDUNDANCY (RPR)

This study proposes a unique architecture, termed Reduced Precision Redundancy" (RPR), as an alternative to the common NMR approach. This approach offers reduced size and power consumption compared to NMR. It also protects against common-mode failures in the algorithm or implementation because of its inherent design diversity. In addition, RPR is relatively easy to implement and can be applied to a broad range of computational tasks.

1. Background

The term RPR has been used in the past by Shanbhag's research team at the University of Illinois to describe their work in reliable low-power digital signal processing (DSP) [55]. Shanbhag's team is concerned with data errors caused when the supply voltage is lowered as a power-saving measure and circuit delays exceed the clock period. This typically leads to large numerical errors, which can be detected and/or corrected by a parallel computation with fewer bits of precision.

While RPR as used in this dissertation is similar in concept to Shanbhag's work, several important differences distinguish this research from any prior efforts. First, whereas the Illinois team has investigated specific DSP applications such as FFT and filtering, this dissertation addresses fault tolerance for general numerical computing problems. Second, their work does not consider SEUs nor faults in FPGA implementations. Furthermore, they assume that the reduced precision module and all comparison/voting circuitry are fault-free. While this is appropriate for their fault model and target hardware, it is insufficient for protecting spacecraft FPGA systems.

Concepts similar to RPR have been widely used in other fields. Littlewood [56] examined software designs, for use in critical safety control systems such as air traffic

control and nuclear power plants, in which "a simple secondary system is used as a back-up to a more complex primary." In these kinds of designs, the more reliable simple system can provide sufficient control, though with less functionality, if the primary system fails for any reason. This architecture is described as "redundancy in which the different components or versions have different levels of trust placed in them."

There are numerous variations on this complex/simple redundancy scheme in the software engineering world, known variously as: primary/backup, primary/alternate and mandatory/optional. Implementations of this scheme include spatially redundant, temporally redundant and combined approaches. In hard real-time systems, this approach can ensure that critical functions are completed according to their stringent schedule. These schemes involve hard tasks that must meet the defined schedule and soft tasks that provide enhanced performance but are not held to the same strict schedule. In real-time operating systems with a single microprocessor, the designs use temporal redundancy by executing the primary and alternate tasks in a certain time order based on an error checking formula [57], [58]. This type of architecture can provide protection against faults in the data or algorithm as well as uncertainties in processor execution time. With a multi-processor architecture, spatial redundancy can provide additional design flexibility and enhanced reliability.

Liu and Han [59], [60] propose similar concepts that they call "imprecise computation" and "performance polymorphism" for balancing reliability and performance in real-time operating systems. Liu [59] discusses imprecise computation for real-time systems in order to gain fault tolerance and handle processor workload uncertainties. In such systems, algorithms must be designed such that intermediate results are available while computing the full precision result. These intermediate results must be monotonically increasing in precision so that the output increases in precision as long as the algorithm is allowed to run. Thus, precision can be dynamically adjusted to balance throughput and checkpoint-based fault-tolerance. Han looks more specifically at scheduling algorithms for optimizing performance using primary (high precision) and alternate (low precision) tasks [60].

While these concepts are common in the software engineering community, there is very little discussion in the literature regarding their utility for FPGA fault tolerance.

The most relevant work, by Kakarla and Katkoori [61], uses "partial evaluation" to build a TMR-like SEU-tolerant design. Their design provides SEU immunity through a unique approach of constructing "reduced circuits" in a spatially redundant configuration. These reduced logic circuits are simplified from the original circuit by determining the most probable value for each signal. A signal with a high probability of being either 0 or 1 is rounded to a constant value and the circuit is reduced. In addition, temporal redundancy is implemented on the full circuit to cover instances when the input data differs from the predicted values. Based upon the actual input data, the system dynamically chooses whether to use the results from the spatially redundant or temporally redundant circuits. One limitation with this approach is the increased latency through the temporally redundant path. Another problem for an FPGA implementation is the vulnerability to configuration faults within the original circuit when low-probability input vectors are received.

2. Architecture Description

RPR is an efficient fault-tolerant architecture that improves the reliability of complex calculations by utilizing approximate solutions from relatively small redundant elements. RPR provides many of the same benefits as TMR, such as error detection, error masking, fault location and ease of implementation. In contrast to TMR, this architecture consumes fewer resources and may be considerably more efficient for many applications. The main drawback to RPR is that its error masking capability can only guarantee an approximate solution.

The concept for "reduced precision redundancy" (RPR) is shown in the figure below. Notice the similarity to the pyramid in Figure 2.4. The bottom level consists of a computation unit that operates at the maximum precision needed for the intended application; thus this unit is called the exact module. In the middle are 2 or more units that perform the same basic function as the exact module, though with less precision; thus they are called approximate modules. At the top-level is a voting unit that decides which of these several results are most likely correct and should be provided as the output data.

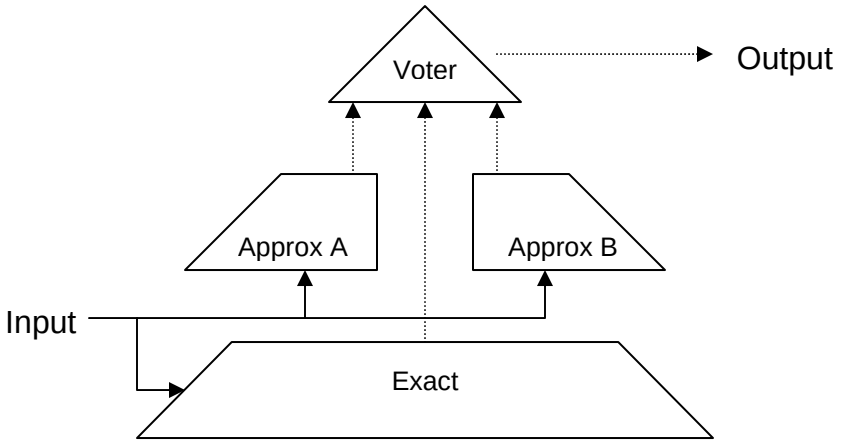


Figure 2.7 Simple RPR Architecture

This architecture is considerably more efficient than TMR if it provides acceptable performance. A major contribution of this research is the demonstration that in many situations RPR performs very well. This dissertation quantifies RPR performance according to computational throughput, latency, accuracy and reliability. In benign conditions, RPR provides performance equal to TMR designs. RPR may be less accurate or less reliable than TMR in high radiation environments, but its overall efficiency is better because it consumes fewer resources. As a general rule, the efficiency of a design can be assessed by its physical size – in an FPGA this can be measured by the fraction of resources used as given by vendor-specific parameters such as slice count and I/O pin count. Power consumption is often proportional to area, though detailed analysis is needed to prove which of various competing designs use less power. If the RPR version of a design is much smaller than a TMR version, it should offer significant power savings.

The resource savings of an RPR design depend heavily on the level of precision needed in the redundant modules. With the wide range of possibilities for implementing the approximate modules, there is a nearly continuous range of possible resource utilization for a design. At one end of the spectrum are designs with very coarse approximations, which utilize extremely small approximate modules. At the other end of

the spectrum are designs that require extremely high precision even in the redundant modules. The limiting case at this end would be a full TMR solution. Chapter IV addresses these issues in more detail.

3. Applying RPR to Computational Problems

In this dissertation, computational problems and algorithms suitable for RPR implementation are termed "Class A" problems. Conversely, problems and algorithms that are not suitable for RPR implementation are termed "Class B" problems. Most numerical computations are Class A since digital systems must represent real numbers with finite precision, introducing some degree of approximation even in the full precision situation. It is a matter of implementation and desired performance that determines the level of precision used for representing the numbers. Other types of problems may also be Class A, such as complex state machines that compute control signals of varying complexity and criticality. Furthermore, all design specifications should be carefully scrutinized, since it is common for system designers to over-specify portions of a design. If a component spec calls for unnecessarily high precision, perhaps the spec can be relaxed or the component's reliability can be improved with lower-precision redundant units.

a. Class A Problems

For most Class A problems, a gradient of importance exists amongst the computed bits. The figure below shows a hypothetical distribution curve where various bits are more or less significant. This figure suggests that more effort should be spent on improving the reliability of the most important data elements and less effort given to the least important elements. This concept is directly applicable to numerical functions since they produce results with easily identified more-important and less-important data elements. In a fixed-point number system the most important data is typically the Most Significant Bit (MSB), which carries the most weight, and the least important data is the Least Significant Bit (LSB) which carries the least weight. Similarly, in a floating-point number system the exponent field is typically the most important and the mantissa (or significand) field is the least important.

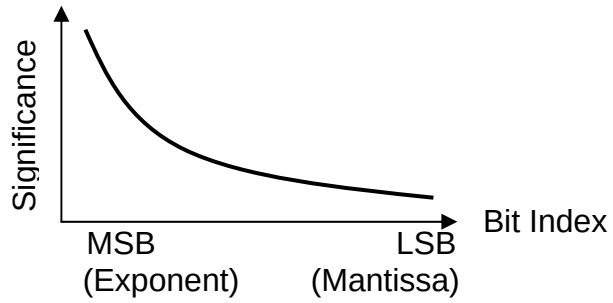


Figure 2.8 Bit Significance Distribution Curve

Addition and multiplication are simple examples of the kinds of computational problems appropriate for RPR. In a fixed-point number representation, an approximate addition can be produced by only including enough bits, starting at the MSB, to meet the minimum acceptable precision. (Keep in mind that for RPR to be useful, the minimum acceptable precision must be appreciably less than the precision needed for optimal performance.) For example, a lazy bookkeeper might balance his checkbook by only adding/subtracting whole dollar amounts – the results will probably be close enough to avoid major financial disaster.

Next, consider the task of multiplying floating-point numbers. More harm can come from errors in the sign and exponent calculations than in the mantissa calculation. Thus, an approximate solution might involve checking the signs and adding the exponents – therefore eliminating multiplication of mantissas and any necessary pre/post-shifting. This type of "order of magnitude" calculation is common in science and engineering when performing initial estimates and "sanity checks."

The tables below illustrate these basic techniques with some simple error-bounding (upper and lower bounds) and bias-removing schemes. Removing bias for addition might involve adding a value of ½ for each element being added, since this is the best estimate of the discarded fractional portion of each entry. For the floating-point multiplication example, note that even the "exact" solution is not really exact since the correct result should be 7,448 instead of 7,424. This, of course, is because of the limited word-length allowed in this case – as is true in most real-world designs. Even with infinitely precise internal calculations, the output cannot represent the number 7,448 with only 4 fractional mantissa bits. Thus "approximate" solutions are inherent in this

calculation. In order to remove bias in multiplication, we could estimate each entry's mantissa as 1.5_{10} and precalculate a best guess for the resultant mantissa based on how many entries are multiplied.

	Unsigned binary	Decimal
A	**01001.**001	9.125
B	**00110.**101	6.625
C	**00100.**110	4.75
D	**00011.**010	3.25
Exact sum = A+B+C+D	**10111.**110	23.75
Integer sum (low)	**10110.**xxx	22
Integer sum + 4*½ (mid)	**11000.**xxx	24
Integer sum + 4*1 (high)	**11010.**xxx	26

Table 2.1 Approximation Methods for 4-Input Sum Calculation

	Floating-point binary sign | exp | mantissa	Decimal Equivalent
A	+ | +**0110** | 1.0011	$1.1875*2^6 = 76$
B	+ | +**0011** | 1.1100	$1.75*2^3 = 14$
C	- | +**0010** | 1.1100	$1.75*2^2 = 7$
"Exact" product =A*B*C	- | +**1100** | 1.1101	$1.8125*2^{12} = 7{,}424$
Exp sum * 1^3 (low)	- | +**1011** | 1.xxxx	$2^{11} = 2{,}048$
Exp sum * 1.5^3 (mid)	- | +**1100** | 1.1011	$1.6875*2^{12} = 6{,}912$
Exp sum * 2^3 (high)	- | +**1110** | 1.xxxx	$2^{14} = 16{,}384$

Table 2.2 Approximation Methods for 3-Input Product Calculation

b. Class B Problems

There are several ways for a problem to fall under the Class B category. First, the full precision calculation of a function may be the simplest solution. Solutions to some problems are either correct or incorrect, with no possibility of being approximately correct. These kinds of problems frequently exhibit no gradient of importance among the output bits, as discussed in the previous section. Also, some problems are only solvable using a single algorithm or method. Most logic functions (such as the vector operations AND, OR, NOT, etc.) fall under the Class B category as there is no meaningful way of describing an approximate solution. Peterson and Rabin [62] proved that for all non-trivial logic functions, except XOR and its complement, the only means of error checking is complete duplication. XOR is a special case because parity bits can be used to detect

and/or correct errors in the XOR computation. For the other logic functions, the simplest circuit for detection of errors requires a coding scheme and checking operation identical to the operation being checked.

Second, even if an approximate solution can be found, it may be insufficient to adequately protect the system against faults. Third, a particular problem or algorithm may be Class B for practical reasons such as size, speed and power. For example, if the only possible reduced precision solution requires more complicated voting circuits than TMR, then RPR is not a wise choice.

Another way of looking at Class A/B problems is to consider the possibility of catastrophic failure. Generally speaking, systems with well-designed feedback mechanisms can tolerate some degree of imprecision/error/noise. However, systems with positive feedback or open-loop systems may not be able to recover from even a single error. If even small errors can lead to catastrophic failure or an unrecoverable state in which complete system shutdown is needed, the task is Class B. Many state machines have certain undefined states that, although unexpected in normal operations, are possible in an SEU environment. In a poorly designed state machine these possibilites may not have been considered and the system may never recover on its own from such a situation. For example, consider a microprocessor that is performing an interrupt service routine. If instead of reading an instruction from the interrupt handler code, the processor unintentionally writes into the memory space holding the code, the processor may get stuck processing invalid instructions until a complete reboot is performed.

c. Approximate Solutions

While it is difficult to give precise definitions of Class A and Class B problems, it is possible to describe whether or not approximate solutions can be found for a particular problem. The following discussion considers only whether RPR is possible, not whether it is advantageous. Since the goal of fault-tolerant computing is to ensure correct data output, it is important to understand how functions act as mappings from inputs to outputs. Some mathematical definitions related to functions help this

discussion. The domain is the set of all input values to the function. The codomain is the set of all *possible* output values from the function, whereas the range is the set of all *actual* output values.

The distinction between "codomain" and "range" is especially significant with regard to fault-tolerant FPGA designs. Since SEUs can cause a function to malfunction in perverse ways (adders can become subtractors, etc.), it is important to recognize that an erroneous output value could exist anywhere in the codomain, not only in the expected range. For example, an algorithm that adds a series of positive integers is expected to produce outputs in the range of positive integers. Yet a misbehaving circuit might output negative numbers. A fault-tolerant design should consider the potential effects from such miscalculations and provide adequate detection/mitigation for them.

Functions are often thought of as "mapping" inputs to outputs and can be visualized as directed arcs linking domain members with codomain members. Such visualizations can help determine relationships between various domain and codomain members. For example, consider the logical function AND for input vectors of up to 3 elements, depicted in the figure below. Since the output from this function can only take on two values, TRUE or FALSE, numerous input values produce the same outcome. Can the function be simplified to take advantage of the fact that the correct outputs can be reached using less input information? Intuitively we know that the answer is no. As shown highlighted in the figure, if only the leftmost bit is considered by the function, the wrong conclusion will be made 3/8 of the time. If only the leftmost 2 bits are considered, the wrong result is produced 1/8 of the time. All 3 bits of input are necessary and no simplification is possible. Therefore the logical function AND is a Class B problem.

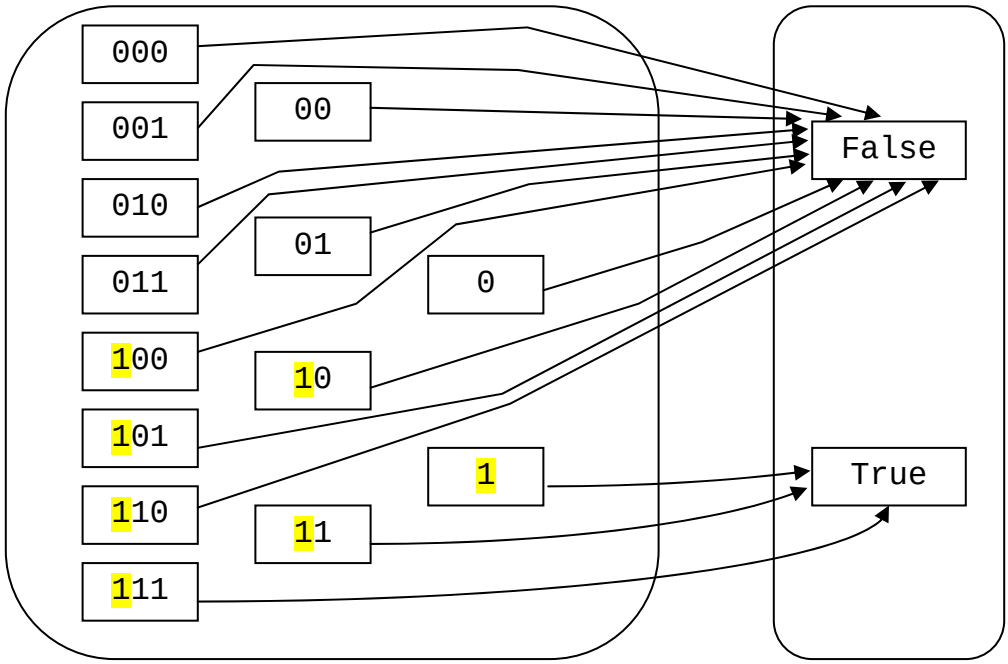


Figure 2.9 AND Function Map for 3-Bit, 2-Bit and 1-Bit Input Vectors

This graph-based technique can be expanded to include more general input and output relationships. The figure below shows input vectors of various lengths (as indicated by the number of contiguous empty boxes) and several output points. By grouping inputs and/or outputs in some way, it may be possible to find relationships permiting simplification of the function. Input groupings may be related to positional features of the vectors, as discussed in the AND example, or numerical ordering. Output groups may be defined precisely (such as all values within a narrow range) or broadly (such as all positive values). Furthermore, a particular output value may belong to multiple groups. An additional feature shown in the figure is that the graph can include multiple distinct functions that perform the same basic operation. In this hypothetical example, a different mapping function is assumed for each input vector size (as shown by distinct linetypes).

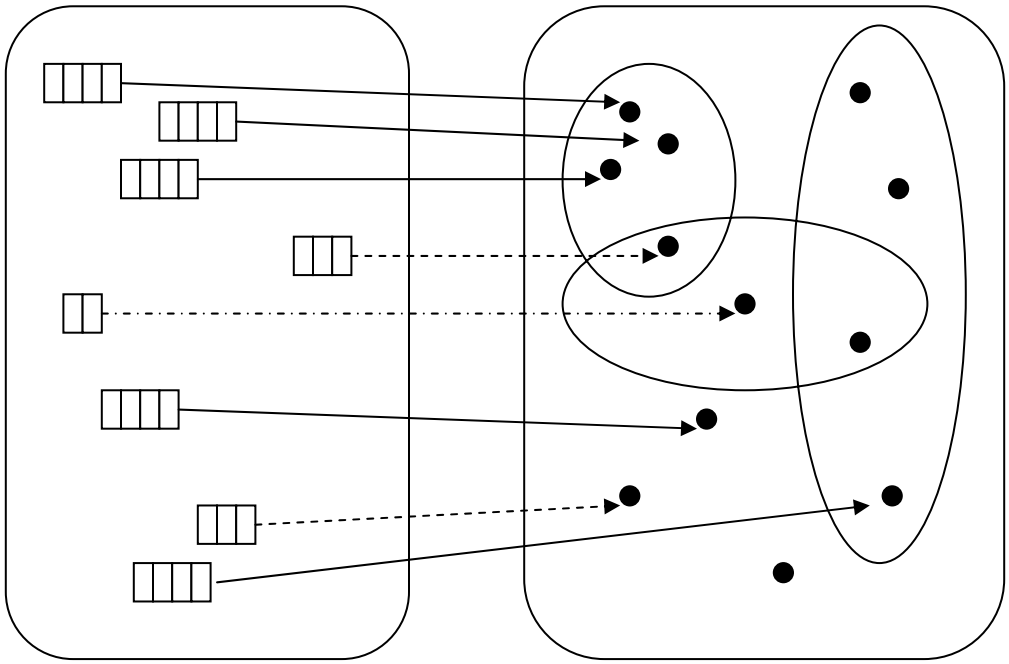


Figure 2.10 Hypothetical Multi-Function Map

This abstract description can also be described as “clustering.” Clusters correspond to the groups defined above, with the additional feature that members of a cluster are more closely related to one another than to any other cluster. If each cluster of input vectors maps to a unique cluster of output vectors, there is potential for simplification through approximation. If, however, there is cross-over between input and output clusters, such a simplification is not possible. If the function map shows clustering, approximation *may* be possible, but is not guaranteed. Clustering is a necessary, but not sufficient, condition for Class A. Functions that exhibit no clustering or clustering that cannot be translated into a simplified function are Class B problems.

Another simple example demonstrates how clustering can reveal potential simplifications. In this example, the function is a simple conversion from a binary number system to decimal. The range for the decimal values is 0 to 7, thus 3 binary digits are needed to precisely map to decimal integers. If the computing system is limited to only 2 binary digits, then it must decide how to interpret the binary values. Assuming the system is required to produce integer values, the value “00x” in the figure could map to either 0_{10} or 1_{10}. Input clusters can be defined by the 2 MSBs {00x, 01x, 10x, 11x}

and output clusters can be defined as the intervals {0-1, 2-3, 4-5, 6-7}. Each input cluster maps to a unique output cluster, so the function can be simplified.

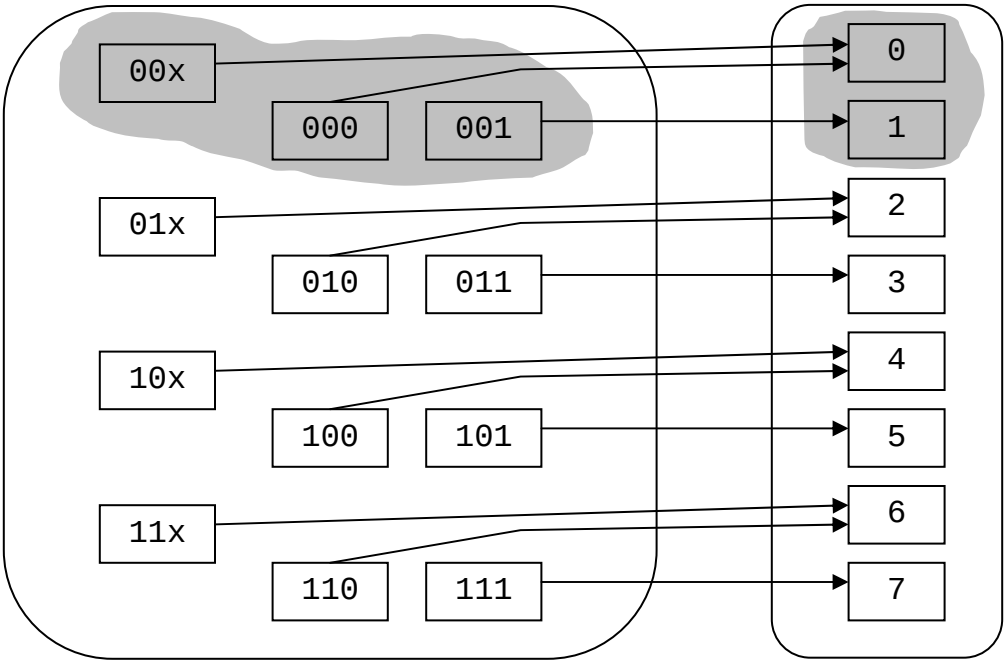


Figure 2.11 Clustering for 2-Bit and 3-Bit Representations of Integers

Approximate solutions can be more easily identified when the problem is defined at the proper level. Although the AND function itself is Class B, it may be part of a larger and more complex algorithm, which at a higher level of abstraction could be deemed Class A. Peterson and Rabin [62] also observed this important distinction and noted that "an adder can be constructed of 'and,' 'or,' and 'not' devices which cannot be simply checked, and yet the addition operation as a whole can be checked quite well without complete duplication."

d. RPR Suitability

The process for determining if a task is amenable to RPR is summarized in the following flowchart. The figure below outlines the steps for determining whether a problem is Class A or B. When using this flowchart, both the algorithm and its intended application must be considered together. A given algorithm may be Class A in some

applications but Class B in others. Similarly, a problem might have several possible solutions with varying precision, but the higher-level system may require full precision constantly.

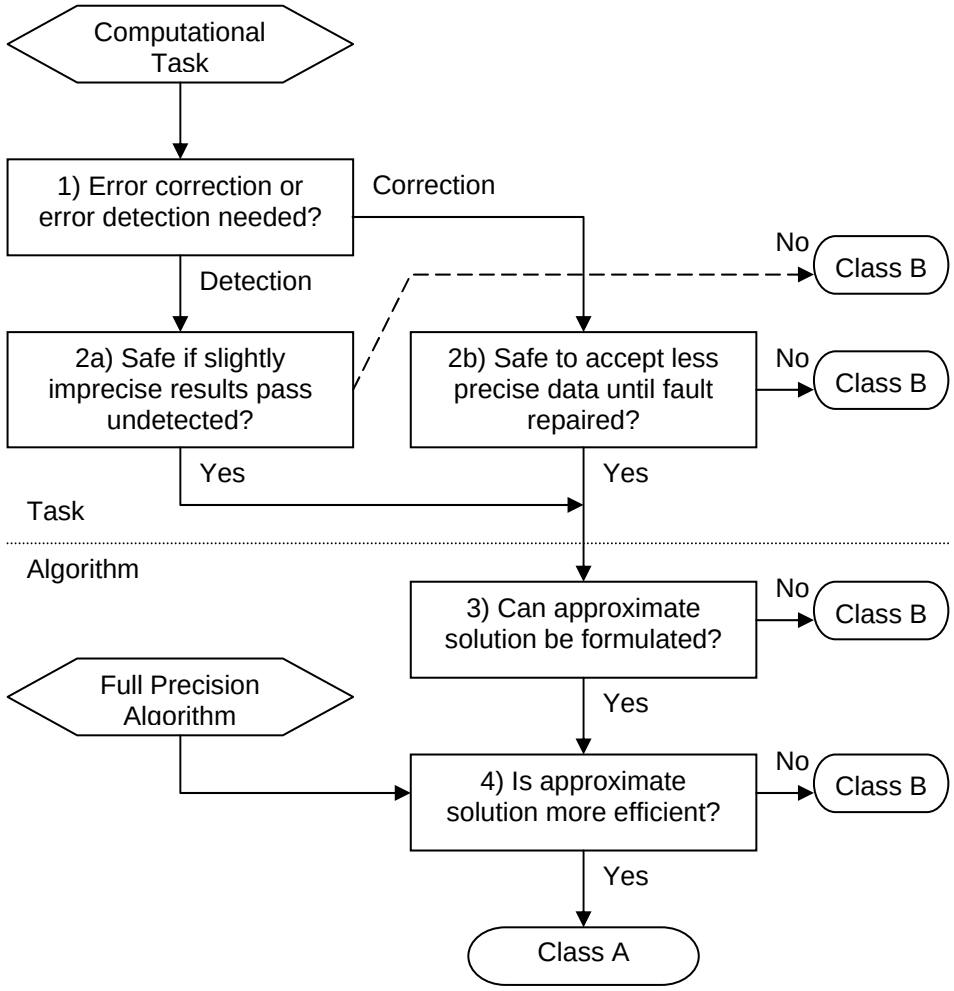


Figure 2.12 RPR Suitability Flowchart

This process begins with a thorough understanding of the desired computational task or problem. Assuming that the system requires some degree of fault tolerance, the first step in the flowchart determines whether error detection or correction is necessary. If the system requires accurate data on a continuous basis, then error correction or masking may be needed. For example, losing a few data frames on an encrypted link may cause unrecoverable corruption of the datastream. In less stringent designs, error detection may be sufficient. For example, a video transmission with

periodic resynching might only require error detection since losing a few video frames to bad data is usually not catastrophic.

If error detection is sufficient, step 2a determines whether full precision error checking is needed or if error bounds checking is adequate. With an RPR design using upper and lower bounds checking, faults in the exact module could cause it to produce slightly imprecise results that the voter would deem acceptable. Such errors would be undetected if they are less than the threshold of the upper/lower bounds checks. If the system can tolerate these undetected inaccuracies, RPR may be appropriate. Otherwise, exact error checking is required – full duplication or TMR is needed – and the task is Class B.

If step 1 calls for error correction, step 2b addresses the safety of the system if the precise calculation is corrupted. RPR can provide fault masking by using output from one of the low precision modules when faults corrupt the high precision results. If the cause is a persistent configuration fault in the FPGA device, this situation will exist until the fault is scrubbed from the circuit. If the cause is a transient data fault, the error may exist for as short as a single clock cycle. In either case, the system's response to imprecise data must be assessed. If imprecise "noisy" results can lead to catastrophic failure, the task is Class B.

The next two steps relate to the specific algorithm(s) selected to perform the required task. Step 3 considers whether it is *possible* to build a reduced precision circuit. If such a circuit is impossible or impracticable, the algorithm is Class B. This was discussed at length in the last couple of sections. The baseline design for the full precision algorithm is shown as an input to step 4, which looks at whether the reduced precision design provides any substantial benefit. If the answers to steps 3 and 4 are affirmative, RPR is a good candidate for achieving fault tolerance in a design and the task is Class A. The next section provides examples to help illustrate this selection process.

e. Examples

Consider a satellite's solar panel servo-control system. Satellites in low-earth orbit (LEO) spend roughly half the time in sunlight and the other half in darkness. While in sunlight, the articulating solar panels are oriented towards the sun to maximize

power output. In this hypothetical example, the designers want to conserve energy while in Earth shadow, so the solar panel control system will not track the sun vector through darkness. Instead, once the satellite enters darkness, it calculates a minimum-energy solution to position the solar panels in the optimum direction for when the spacecraft reenters sunlight. Fault tolerance in this application should ensure the solar panels face in the general direction of the sun upon exiting Earth's shadow. A miscalculation that directs them away from the sun could be very damaging (loss of power could lead to loss of spacecraft operation).

The algorithm for this calculation is likely to be quite complex. However, there are simple approximate solutions, such as using the position calculation from the previous orbit as an approximation and error-checking value. Error detection may be adequate for a LEO satellite, which spends roughly 40 minutes in darkness per orbit, allowing plenty of time to recalculate if an error in the detailed calculation is detected. Since the approximate calculation can be performed with very little circuitry, RPR may be appropriate for this problem.

As another example, consider a satellite attitude control algorithm designed to keep the satellite pointed in a desired direction. Inputs include various parameters such as position, velocity, current orientation, desired orientation, etc. Outputs consist of commands to thrusters and reaction wheels. The basic design specification includes the computational precision needed to meet the satellite pointing accuracy requirement. At steps 1 and 2 in the flowchart of Figure 2.12, we find that the system cannot safely stall long enough for a fault to be repaired. Complete reconfiguration of an FPGA may take tens of milliseconds, allowing the satellite to drift dangerously far from the required position. Therefore, error correction/masking is needed. However, small deviations from the desired position/orientation can be accommodated by the natural feedback in the control system, so occasional small perturbations are acceptable. Quantifying the frequency and amount of acceptable perturbation requires careful scrutiny of the design. At step 3 in the flowchart, there are many approximate solutions, ranging from lower precision numerical calculations to simply ignoring some input parameters. Finally, at step 4 we anticipate that the simpler

control algorithms will be smaller and consume fewer resources than the more exact algorithm. Therefore, we conclude that RPR is a viable option for this design.

Data compression algorithms provide interesting examples for considering the applicability of RPR. Some data compression tasks are Class A, while others are Class B. Lossy compression techniques, such as JPEG, can usually tolerate some imprecision. One of the standard JPEG compression techniques performs numerous multiplications and additions as part of calculating the discrete cosine transform (DCT) for each subarray in an image [63]. A transient error during one of these calculations leads to poor image reconstruction in only a small portion of the total image, leaving the rest of the image intact. Furthermore, even faults causing reduced precision across an entire image are generally tolerable, as they degrade image quality but are not catastrophic to the image processing system. Thus the DCT calculation for JPEG image compression is Class A. (Chapter VIII looks at this example in more detail.)

There are numerous lossless compression techniques, with widely varying methods of compressing data. One of the most common lossless compression codes is the Lempel-Ziv code (a standard part of ZIP programs). This code is based on dynamically building a dictionary of common "phrases." Compression is possible when addressing uses fewer bits than the phrases themselves. The program compresses data by addressing the dictionary entry that contains the phrase, rather than the phrase itself. Lempel-Ziv is a Class B algorithm since a corruption of the dictionary can lead to complete failure of all subsequent phrase/address calculations [35]. Other lossless compression techniques may be amenable to RPR application. For example, Huffman coding, which is used in some facsimile transmission and JPEG standards, is based on predicted symbol statistics [64]. Imprecise calculation of these symbol statistics may cause graceful degradation as in DCT, rather than abrupt failure as in Lempel-Ziv.

f. Applying RPR to non-FPGA Systems

While this dissertation focuses on FPGA designs, the RPR design philosophy can be extended to other technologies. As discussed in Section 1 above, RPR is similar to techniques used in other fields. Many of the design parameters considered in this dissertation (area, speed, reliability, etc.) are directly applicable to any digital

computing technology. However, RPR is especially suitable for FPGAs because its redundancy structure addresses the large range of possible faults, including configuration faults, that affect circuit behavior. For traditional digital systems implemented on non-volatile circuits, transient faults affect only data values and leave circuit function intact. For these systems, many other fault tolerance approaches, such as data coding, may be appropriate. Nonetheless, because of RPR's scalability and ease of implementation, it offers a unique and competitive option for fault-tolerant designs.

5. Flexible Precision Computation

The preceding sections focused mainly on determining whether it is possible to apply RPR to various computational problems. A higher-level system perspective is necessary to determine whether RPR is practical in a certain application – steps 2a and 2b in the flowchart of Figure 2.12 address this issue. A key performance parameter of any numerical computation is the precision of its output. Precision is typically measured by the number of bits used to represent the whole and fractional parts of the true numerical value. For example, a 10-bit fractional binary number can represent real numbers to within $+/- 2^{-11}=0.000488$ of their true value.

Digital design specifications generally include the precision required for a given application. Designers often use a worst-case scenario that requires maximum precision to determine these requirements. A system built to these conservative specifications will provide the necessary performance under all conditions. However, many computing applications can operate at lower precision for short periods of time with minimal or no adverse effect. The term "flexible precision" refers to designs that take advantage of this possibility by adapting to meet variable precision needs.

Flexible precision computation is a viable technique for many applications. For computer graphics processing, such as in video gaming, the fidelity of image rendering can adapt according to scene dynamics, display device properties and viewer preferences. Furthermore, the limits of human visual perception can be exploited to simplify computations that produce results of higher precision than the human eye can perceive [65].

Control systems is another field where flexible precision seems promising. For example, a satellite's attitude control system senses the satellite's position/orientation and commands thrusters and reaction wheels to achieve a desired end-state. Such systems must have a very short response time and be very reliable. In this situation, it is preferable for the control system to constantly provide accurate but less precise commands than to occasionally fail abruptly and issue inaccurate and possibly detrimental commands.

As a more specific example, imagine a reconnaissance satellite that images specific locations on the earth's surface. The satellite must be oriented such that the imaging sensors can view the approximate ground locations. This general orientation control is sometimes called "coarse alignment." Typically the imaging sensors themselves have at least one feedback control loop to permit finer control for keeping the target image centered in the camera field-of-view. This more precise pointing control is called "fine alignment" and is commonly used on ground-based telescopes with fast steering mirrors [66]. Fine alignment is also used on consumer products, such as video cameras. Many hand-held video cameras offer "image stabilization" features that use optical and/or electronic techniques to compensate for jitter. Ideally, coarse alignments control the reconnaissance satellite so precisely that the fast steering mirror need only compensate for small-amplitude, high-frequency jitter. The inherent redundancy of this arrangement, however, allows for some degree of imprecision in the coarse alignment.

If a system can accommodate flexible precision then it is likely Class A. Algorithms that take advantage of flexible precision, such as the dynamic graphic rendering technique mentioned above, are prime candidates for RPR. In addition, systems that function best with full precision but can safely operate with temporarily reduced precision, can also benefit from RPR's efficiency. Steps 2a and 2b in the flowchart address whether flexible precision can be safely utilized in an RPR structure.

F. VOTER ISSUES

Voting techniques are important to the overall effectiveness of a fault-tolerant design. Descriptions and classifications of a wide variety of voting techniques are provided in [67]. A common NMR voting method is to take the bit-wise majority as the

correct output vector [68]. However, when faults exist in more than one module, this voting style may produce an output that doesn't exactly match any of the pre-voted results. The table below demonstrates this dilemma for 3-modular and 5-modular redundant systems. The bit-wise majority output vector may not even represent a permissible output. Furthermore, in the hypothetical 5-modular example shown, a fault analysis based on the bit-wise majority would conclude there must be faults in all 5 modules. The vector-wise majority implies that only 3 modules are faulty. In this example the vector-wise majority seems superior, since fewer faults is the more likely scenario. Gersting [69] studied alternative voting schemes that partially address these issues.

	3-Modular Redundancy	**5-Modular Redundancy**
Module A result	101100	101100
Module B result	011111	011111
Module C result	100011	100011
Module D result		101100
Module E result		010101
Bit-wise majority	**101111**	**101101**
Vector-wise majority	**??????**	**101100**

Table 2.3 Hypothetical Outputs from 3-Modular and 5-Modular Redundancy

In some situations an inexact voting method is necessary. For example, when comparing the outputs from redundant analog-to-digital converters, it is quite likely that the various outputs will not match exactly due to inherent noise in the system. Error detection in this scenario must include some error thresholds in order to distinguish between truly faulty modules and simply normal data variability. Some of the issues with inexact voting are the determination of appropriate thresholds and the increased complexity of inexact voting circuits [67]. Lorczak [70] proposes a generalized median voter for addressing some of the difficulties with inexact voting. In addition, Lorczak provides several examples demonstrating situations when various alternative voting methods are optimal.

These issues of inexact voting are important in an RPR implementation. In considering the simple RPR architecture shown in Figure 2.7, it is clear that the voter must decide whether the exact solution is acceptable based only upon information from inexact redundant modules. With a binary pass/fail criteria, the voting methodology must properly merge the data with the understanding that the various modules will not match one another exactly. The example in the table below demonstrates this dilemma.

	Binary Two's complement	Decimal
8-bit "exact"	**0011**1001	57
4-bit w/ truncate	**0011**xxxx	48
4-bit w/ rounding	**0100**xxxx	64

Table 2.4 Example Relationship Between Alternative Numerical Approximations

The upper four bits of the exact solution match the approximate solution calculated by truncation, but not the calculation based on rounding. A sophisticated voter could use either approximation technique and apply an error bound before declaring the exact solution correct or incorrect, but this would require a larger and more complex voter circuit. Using spatial redundancy in the middle layer can keep the voter as simple as possible. Two approximate modules, for example, could calculate upper- and lower-bounds that differ by one in their LSB position. Then the voter need only compare the most significant bits to determine if the exact solution matches one of the approximate values and is therefore within the error bounds. (Chapter VIII discusses these issues in more detail.) However, this approach requires the approximate modules to be highly reliable, since the voting is essentially a 2-out-of-2 pass criterion. To increase the reliability of the approximations and voting, further redundancy within the middle and top layers of Figure 2.7 may be appropriate, since duplication of these smaller modules can be simpler and smaller than duplication of the exact module.

G. SUMMARY

This chapter has developed RPR as a new approach for achieving SEU tolerance in FPGAs. Like other fault tolerant methods, RPR requires error checker and voter

components that have higher reliability than the circuits they are protecting. In FPGA designs this can be achieved if the checker/voter modules are significantly smaller than the primary computation module, as smaller circuits should be less likely to suffer SEUs than large circuits. Chapters VI and VII present data confirming this hypothesis. The other main advantage of RPR is that the relatively small redundant calculation circuits require less area and power than the redundant circuits in TMR. Chapter III discusses FPGA power reduction in more detail and Chapter VI provides data confirming RPR's power advantage. Finally, Chapter VIII revisits several RPR implementation ideas introduced here, including proper upper/lower bounding, the use of lookup tables, and the effect of imprecise calculations.

THIS PAGE INTENTIONALLY LEFT BLANK

III. POWER SAVINGS TECHNIQUES

A. POWER EFFICIENCY FOR FPGA DESIGNS

The tremendous demand for energy efficient designs in the commercial market has spurred development of power-aware circuits and system architectures, especially in the last decade. There are many approaches for reducing power consumption in digital circuits. However, some of these approaches are not viable for systems using FPGA devices. Furthermore, some approaches degrade the radiation and/or fault tolerance of a system. This chapter describes the most promising methods for reducing power consumption in FPGA systems, while considering their impact upon the system's fault tolerance.

In addition, this chapter discusses the potential power savings of an RPR architecture over standard TMR and other fault tolerance methods. Once a design for the full-precision circuit is established, the engineer must choose what redundancy method(s) to use for protecting the circuit. Although there are many intriguing options for minimizing power consumption, a principal means of achieving power efficiency is through reducing the physical size of a circuit. A key feature of RPR is that it requires much less chip area than TMR. Therefore, RPR's reduction in circuit size offers significantly reduced power compared to TMR. Though the total area and power savings depend on the particular circuit and precision requirements, the RPR circuits built for this research use between 1/3 and 1/2 the circuit area as the equivalent TMR designs. As is shown in Chapter VI, effective RPR designs can use less than half the power of TMR and only slightly more power than the original unprotected circuit.

B. BACKGROUND

Power consumption within CMOS-based (Complementary Metal-Oxide-Semiconductor) FPGA circuits can be divided into two categories: static and dynamic [30]. The total power consumed in a device is simply the sum of these two components:

$$P_{total} = P_{static} + P_{dynamic} \tag{3.1}$$

Static power , given by $P_{static} = V_{supply} \times I_{static}$, is determined by all electrical current that flows when a circuit is "powered up" and ready to perform useful functions. Static power does not depend on whether the circuit is actually processing data, which is usually determined by a clocking signal. Ideally, electronic devices would consume zero static power. However, appreciable static power can arise from device imperfections, biasing configurations, and basic physical design characteristics. In older technologies such as Resistor-Transistor Logic, the resistors used as bias elements consumed considerable static power. Transistor leakage current is the most significant component of static power in modern CMOS technology. In current technology, static power is typically much smaller than dynamic power and is often ignored in general CMOS circuits [38].

Dynamic power is consumed when signal transitions occur in a circuit. When a signal changes from high-to-low or vice versa, current flows in order to charge or discharge the effective capacitance of that signal node. This effective capacitance includes the actual capacitance at the node, short-circuit effects during transitions, and ohmic losses from non-ideal device properties.

The basic formulas for dynamic power in CMOS circuits, such as FPGAs, are given in Equation 3.2 below. The first formula gives the average power consumption for a single node within a circuit, while the second formula simply sums the power over the entire circuit. In this context, a node might be defined at the transistor level or at the logic gate level. Either definition can be used so long as the node transition activity factor α and effective load capacitance C are properly determined.

$$P_{dyn-node} = \alpha f C V^2$$
$$P_{dyn-total} = \sum_{i=1}^{N} \alpha_i f C_i V^2 \qquad (3.2)$$

where α node transition activity
f clock frequency
C load capacitance
V power supply voltage
N total number of circuit nodes

1. Relative Contributions of Static and Dynamic Power

The percentage of power attributed to static and dynamic terms depends strongly on the device technology and the specific circuit under investigation. Dynamic power dominates for typical CMOS devices such as microprocessors and FPGAs [10], [71], [72]. However, modern FPGAs have routing and control structures that draw substantial static power, mostly through transistor leakage current [9]. Guo [73] notes that, depending on factors such as design configuration and operating frequency, "static power is between 5%-20% of the total power dissipation in Virtex-II." As FPGA circuit densities continue to increase, static power will increase proportionally with the total transistor count. Furthermore, CMOS technology is evolving towards thinner gate oxides and lower threshold voltages, both of which increase leakage current [74]. In the future, static power may become dominant in FPGAs. Thus, static power should not be ignored in FPGA designs.

Whether static or dynamic power dominates is important for power optimization in circuit designs. Where static power is dominant, emphasis should be placed on circuit designs that use fewer transistors or that can disable inactive circuit elements. When dynamic power dominates, designs with fewer signal transitions are most beneficial. The following two sections discuss ways to reduce static and dynamic power.

2. Reducing Static Power

Static power is essentially a fixed quantity based on the semiconductor process used to build the device. Therefore, there are limited ways in which a circuit designer can reduce this term in the power equation. Designs using COTS products, such as most FPGA devices, have little or no opportunity for influencing static power consumption. However, ASIC or custom FPGA designs can utilize the techniques described below for minimizing this component of power.

One way of minimizing static power consumption is to attack the fundamental physical imperfections that give rise to this power term. Various semiconductor enhancements have been pursued to achieve this goal. The use of CMOS devices is a major improvement over older circuit technologies. The counter-balancing NMOS and PMOS transistors in a CMOS circuit provide fast signal transitioning and very low static

current draw. Nearly zero static current flows because in both the 0 and 1 states, one of the two transistors (PMOS or NMOS) connecting V_{supply} and ground is in cutoff mode. MOS transistors act as nearly perfect switches in cutoff mode by eliminating the conductive "channel" between the transistor's source and drain. Virtually all modern semiconductor devices, including FPGAs, use CMOS technology.

Given that most FPGA circuitry is based on CMOS devices, it is important to focus on approaches that address the unique issues of this technology. Although CMOS devices act as excellent switches, they are not perfect. Even when MOSFETs are in cutoff, inevitably some leakage current flows. This leakage current has two components, gate and sub-threshold leakage. Gate leakage can only be improved by modifying the gate oxide material and dimensions; such changes typically degrade transistor performance [74]. Minimizing sub-threshold leakage requires the use of transistors with higher threshold voltages [74]. This has two counter-acting effects upon radiation tolerance. It degrades radiation tolerance by reducing noise margins (the gate-to-source voltage must be kept high to ensure the transistor remains on), but somewhat improves tolerance by increasing the threshold that transients must overcome in order to upset the node (a transistor that is off must suffer a relatively large gate-to-source voltage spike in order to be switched on) [29].

A more direct way of minimizing static power is simply reducing the transistor count in the circuit. Fewer transistors mean less leakage power. However, in FPGAs the total number of transistors is fixed. Every transistor contributes to the total static power consumption, whether or not it is a part of the functional circuit being implemented on the device. One approach to overcoming this limitation is to include circuitry within the FPGA to force unused portions of the device into low-power standby modes [74]. This technique, widely used in ASICs, determines when specific subcircuits are temporarily inactive and places them into a standby or "sleep state" in which transistor leakage power is reduced. Unfortunately, such features aren't available with the current generation of FPGA devices.

3. Reducing Dynamic Power

Whereas static power in FPGAs is basically a constant value, a circuit designer has control over many parameters affecting dynamic power. Power can be reduced by minimizing any of the terms in Equation 3.2, without increasing the other dynamic or static power terms. This is particularly challenging when certain parameters affect these terms in opposing ways. For example, semiconductor processing changes that permit operation at lower voltage V can lead to higher static power draw since lower threshold transistors are more prone to leakage current [12]. Dynamic power reduction is especially difficult with FPGAs because the devices' electrical characteristics preclude some power saving techniques that are used in VLSI designs. In custom VLSI circuits the designer has control over many parameters, such as transistor dimensions and conductor lengths. These parameters, which affect node capacitance C and signal propagation delay, are fixed in FPGAs. Power consumption in FPGAs is also very sensitive to design placement and routing [9], as this affects the capacitance and transition activity of signal nodes. However, optimizing the complex placement and routing process is challenging.

Large improvements can be made by reducing the power supply voltage, since it is squared in the power equation. This has been exploited extensively over the years. Older CMOS technologies were designed to work with standard TTL 5 V signals, whereas many CMOS devices now use power supply voltages of 1 V and below [30]. Many modern FPGAs use multiple supply voltages within the chip, using the lowest voltages for the "core" logic and higher voltages for I/O components. However, lower voltages can decrease the speed of a circuit [75]. A popular approach called dynamic voltage scaling involves adjusting the voltage supplied to portions of a circuit as the speed and performance requirements fluctuate.

Another fairly simple way of minimizing Equation 3.2 is by lowering the clock frequency. This has a linear affect on the total dynamic power but also negatively impacts system throughput. In order for a slower circuit to complete the same number of calculations as a faster circuit, it must run for a longer time. Though the slower circuit requires less average power, it typically uses more total energy to provide the same

computational service. However, in situations where the throughput requirements change dynamically, a variable frequency clock design may be beneficial.

The next term to address is the capacitance. Semiconductor advances have permitted smaller device dimensions and shorter interconnections, thereby helping to reduce dynamic power with each new generation. In ASIC designs, the layout and length of wires can be optimized to minimize the capacitance of nodes with high toggle rates. However, in FPGAs the distance between components is fixed and therefore the capacitance values are predetermined. Nonetheless, some optimizations are possible with FPGAs. For example, designs can be compiled such that nodes with high toggle rates are placed onto shorter wires [76]. Genetic algorithms have even been applied to this problem, such as in [77] where the mapping process attempts to place frequently changing signals on short, low-capacitance paths within the configurable logic blocks (CLBs) while avoiding paths that transit the relatively high-capacitance interconnect switches outside the CLBs.

The most intriguing solutions for reducing dynamic power involve minimizing signal transitions. Clock gating is effective at preventing signal toggling in situations where portions of a circuit can be temporarily disabled. For example, adding a zero detection/bypass element allows a multiplier circuit to be disabled whenever any input value is zero [37]. Various techniques have been applied to achieve spurious toggle reduction by avoiding calculations that are inconsequential. Guo, et al. applied such a method to an FPGA implementation of the Viterbi decoder and achieved nearly 50% reduction in power [73]. However, clock gating and similar techniques introduce additional clock skew, which limits the maximum clock speed and may increase glitching effects.

Another major cause of dynamic power consumption is glitching. Glitches occur when unequal delay paths cause a signal to toggle several times before settling to the desired value. As shown in Figure 3.1 below for a hypothetical circuit, these delays cause downstream elements to undergo more toggling than a zero-delay model would indicate. In this example, each logic gate is modeled as having a unit-delay and the input vector is assumed to change instantaneously from (0110) to (1101) at time t=0. The longest path passes through three logic elements, so the output is not guaranteed valid

until time t=3. For this example, the output F undergoes one extra low-to-high and one extra high-to-low transition, each of which consumes extra power. This problem is amplified as a circuit's logic depth increases. While unequal delay through different circuit paths is inevitable, glitching can be somewhat controlled. Pipelining, delay equalization buffers [36] and asynchronous enable signals [78] can minimize the propagation length of glitches in order to conserve power.

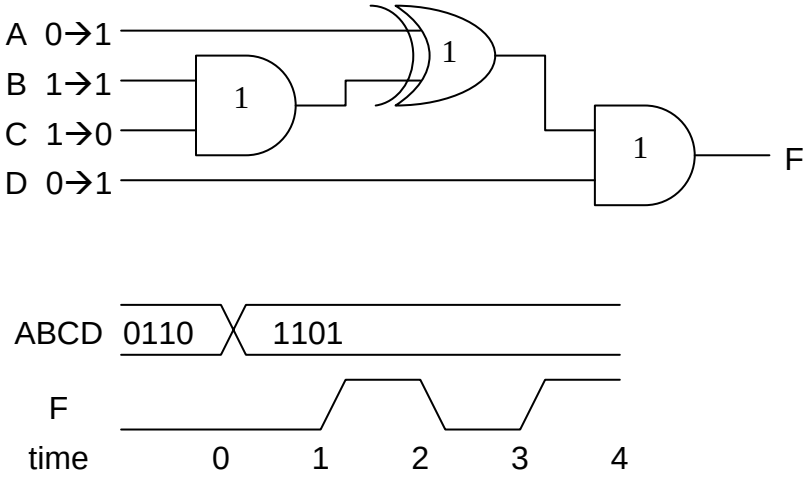


Figure 3.1 Example of Glitching Behavior

Even the choice of number representations can affect power consumption. In a traditional two's complement number system, when values fluctuate between small positive and small negative numbers, many bits must toggle. Signed magnitude number systems or offset value number systems can reduce this toggling and conserve power [79].

Finally, as stated earlier, dynamic power can be reduced by minimizing the number of nodes *N* in a circuit, which is equivalent to reducing the circuit's size. As the summation in Equation 3.2 is performed over fewer nodes, there are fewer contributions to the overall dynamic power. This is the basic premise for reducing power costs in RPR designs. Although larger circuits generally consume more power than smaller circuits, this is not universally true. For example, pipelining increases circuit area, but can significantly reduce glitching power [78]. Thus, assessing the relationship between circuit size and power consumption requires careful analysis. Chapter VI shows that

circuits with similar functionality and structure exhibit a strong correlation between size and power consumption. Using several test circuits computing the same function, the non-pipelined circuits have a roughly linear relationship between size and power. The larger pipelined circuits require considerably more power, but are more efficient in terms of energy usage per calculation.

C. IMPACT OF FAULT TOLERANCE ON POWER USAGE

Although the preceding sections describe power issues that apply to FPGA circuits in general, this dissertation seeks to identify power efficient methods of achieving SEU fault tolerance. Furthermore, this research focuses on design optimizations that can be applied to commercially available FPGAs using current semiconductor technologies. As explained in the previous section, this constraint implies that efforts should focus on reducing dynamic power. If future FPGAs include features such as standby mode switches for reducing leakage current, it will become relevant to address static power as well.

In fault-tolerant designs, overall power consumption is strongly affected by the type and degree of redundancy. Generally speaking, the extensive redundancy structures needed to achieve high levels of fault tolerance consume more power. The main focus of this dissertation is assessing the effectiveness and efficiency of the RPR architecture. It is hypothesized that because RPR designs can be much smaller than TMR designs, the power savings will outweigh the degradation in data precision and/or fault tolerance for many applications.

1. Power Cost of TMR

As the most common approach for providing SEU tolerance in FPGAs, TMR serves as a good baseline for comparing various alternatives. A full TMR implementation includes three complete copies of the functional circuit and a voting mechanism. Thus, one would expect that the power consumption with TMR is more than 3 times that of the unprotected circuit.

Indeed, researchers at Brigham Young University and Los Alamos National Laboratory have confirmed that TMR designs on FPGAs require roughly triple the power

[9]. Their studies included both computer simulations (using precise timing models from ModelSim combined with device models in Xilinx's XPower software) and hardware measurements (using a Virtex test setup called JPower built at BYU). Testing several designs consisting of incrementers, counters, an 8-bit CPU and a QPSK demodulator, they found a 3x-7x increase in power consumption. Much of the reason for this large variability was due to design-placement dependencies. Better power efficiency was observed for compact design layouts, whereas layouts with components placed far apart consumed dramatically more power. Rollins' [9] main conclusion was that with careful design placement, TMR used approximately triple the power.

Their results also indicate that for smaller circuits, and especially the latest generation FPGAs, static power dominates at low operating frequencies. Again, this is because all portions of current FPGAs draw static current whether or not they contribute to the circuit functionality. At low clock frequencies, there is relatively little dynamic activity so dynamic power is less significant. At higher operating frequencies, dynamic power dominates and the latest generation devices with small feature sizes offer better overall power performance because of their smaller geometries.

2. Alternative Solutions

Several researchers have looked at alternatives to TMR that reduce the power cost of fault tolerance. Some of these approaches involve modification of the underlying FPGA architecture, while others are appropriate for use with standard devices. In general, fault tolerance and power efficiency are conflicting goals so designers must strike a balance in the spectrum of design options.

Maheshwari, et al., investigated architectural and circuit-level optimizations for general VLSI circuits in [29]. Since FPGAs are essentially VLSI devices, such optimizations are applicable to the problems investigated in this dissertation. Part of their research compared the effectiveness and efficiency of area-redundant and time-redundant architectures. Their dual modular redundant (DMR) and time redundant designs both yielded a 7x-9x improvement in mean time to failure (MTTF) while consuming 2.3x-2.9x the power of the unprotected circuit. They found that time redundancy uses less power since it only recomputes results that fail an error check. A major concern with applying

their approach to FPGA circuits is the assumption that an error detection unit can be constructed and always operated correctly. For complicated circuits the error detection unit may be very difficult to create and will consume additional power. Furthermore, in a radiation environment the error detector is also susceptible to faults and will not always operate correctly.

Maheshwari also studied various circuit-level modifications, which had much less affect on MTTF and power than the architectural approaches. They found that lower operating voltages degraded MTTF, but improved power dissipation considerably. Using larger sized transistors caused marginal gains in MTTF, but huge increases in power consumption. However, these types of modifications are not possible with currently available FPGA technology. Therefore, circuit-level modifications would be most useful for future generations of FPGA devices.

Others have investigated the use of temporal redundancy to reduce overhead costs of fault tolerance. In [41], a combination of spatial and temporal redundancy is used to reduce area and input/output pin counts, and consequently the power dissipation of a circuit. Their approach is essentially DMR with the addition of delay registers and triplicated voters. Even though they show less area and pin usage than TMR, their approach uses more than twice the area of the unprotected circuit. Therefore, one would expect their design to require roughly twice the power. Furthermore, although this approach can identify which computation unit experiences a transient fault, permanent faults can only be detected and not isolated to a particular module. Thus, this architecture provides much less comprehensive fault tolerance than TMR.

Another way of creating more efficient fault-tolerant designs is to optimize particular circuit structures that occur frequently. Tiwari and Tomko studied the fault tolerance and overhead costs of different FPGA implementations of finite state machines [80]. By implementing state machine functions in an FPGA's synchronous embedded memory blocks (e.g., Virtex BlockRAM) and using a combination of parity bits and internal/external memory scrubbing, they demonstrate lower power than a typical TMR design using triplication of CLB logic and routing. They assume that faults in the routing and other FPGA circuitry are corrected by configuration scrubbing. However, this approach may allow SEU-induced errors to persist for a considerable amount of time.

Furthermore, although the proposed architecture saves size and power, their designs still require between 66% and 92% of the TMR power levels.

3. Power Advantages of RPR

The reduced precision redundancy (RPR) approach presented in Chapter II holds the promise of significantly reduced power consumption compared to TMR and other alternatives. Lower power consumption is enabled in several ways by the RPR architecture. First, the smaller circuits needed for the redundant modules, in general, will use considerably less power than the full precision module. The exact amount of this power reduction will depend on the particular computation being performed and the degree of precision used in the redundant modules. Secondly, the inherent design diversity in the redundant modules permits exploration of additional power optimizations not possible with normal TMR. Since the redundant calculations can be designed differently than the full precision calculation, there is flexibility in applying low power techniques to each module individually. Finally, these power optimizations are easier than in TMR since the redundant modules are smaller and simpler.

An integral part of this research is the demonstration of the potential power savings of an RPR design. Given an original full precision circuit, the designer has several options for the redundant calculations. First, the same methodology can be used and simply scaled to match the data precision desired. The CORDIC algorithm (see Chapter V) is a good example of a circuit that can be easily scaled using the same basic architecture. Second, a table lookup method can be used if the required precision of the approximate calculations is reasonably low (between 8- and 12-bit precision is reasonable for the CFTP experiment). Table lookup is often impractical for the full precision calculation due to the enormous memory requirements, but in the approximate calculation this may be a more reasonable option. Modern FPGAs have significant on-chip memory resources to enable such an approach. Each Virtex FPGA used on the CFTP board has nearly 40 KB of RAM capacity, though 70% of this capacity must be shared with LUT-based logic functions. If the remaining 30% were used for table lookup, two 12-bit addressable 12-bit wide tables could be constructed. Third, the redundant modules can use completely different computation methods than the full precision module. Since the

timing constraints will typically be limited by the full-precision module, the redundant calculations have a relatively generous timing allotment for computing less precise results. This opens many more possibilities for the designer. This diversity also offers protection against certain design flaws that could lead to common-mode failures in TMR.

RPR is expected to show considerable power advantages in many applications. The size reduction of an RPR architecture will, in theory, offer benefits in both static and dynamic power consumption. As a rough estimate, one can assume this power reduction is proportional to the size reduction achieved by using the smaller redundant modules of RPR. When using current FPGA technology, however, static power is a fixed value for a given device size. Thus RPR will improve static power only in cases where a larger TMR design would require using a larger FPGA device. Therefore, this dissertation focuses on techniques to reduce dynamic power using the RPR approach.

Chapter VI quantifies this power reduction for several sample designs using data from high-fidelity power simulations. RPR versions of three different CORDIC algorithm implementations were constructed using between 37% and 46% as much area as the corresponding TMR versions. Power simulations reveal that the RPR circuits require between 37% and 61% of the TMR total power levels. The dynamic power ratios range from 32-52%. These results verify that circuit size is correlated with power consumption, and show that reducing circuit size can effectively decrease power usage.

IV. DEVELOPMENT OF A TOTAL PERFORMANCE METRIC

A. OBJECTIVE

This chapter presents a methodology for assessing the relative merit of diverse solutions to a design problem. In particular, it is suitable for comparing RPR designs to TMR and other approaches. While stressing the importance of fault tolerance, this methodology also accounts for practical design issues such as power consumption and chip area. This method provides designers with a tool for selecting efficient and reliable solutions.

B. CONCEPT

To objectively assess competing design alternatives, a mathematical framework must be established for balancing various design parameters such as power, area, accuracy, and reliability. The typical engineering design process begins with a set of design constraints and focuses on optimizing only one or a few performance metrics. For example, an engineer tasked to design an FFT processing engine is given design specifications (usually max/min values) that dictate board/chip area, power consumption, accuracy, latency, throughput, etc. The engineer then creates a design that meets all the specifications (often right up to the max/min levels) and optimizes the most important performance criteria for the design. The problem with this process is that focusing on only one or a few key criteria, which are often determined subjectively, can obscure potential solutions that optimize the design at a global system level. This process can be improved by employing a more structured mathematical analysis and expanding the number of variables that can be traded off against one another.

The key step in this kind of mathematical analysis is developing honest and understandable cost/benefit functions. This often involves both subjective and objective criteria. Rather than operating independently, a system engineer often works within a team and interacts with colleagues. The expertise and experience brought by these outside sources help formulate cost/benefit criteria that lead to a truly “optimal” system design. By developing performance metrics like those in the following sections, the

system engineer creates a well-documented design process that can be discussed and refined. As the technologies, customers, market competitors, and other factors change, this performance metric tool can help determine the best evolutionary path for a system. The flexible and dynamic nature of FPGAs make them particularly amenable to an evolutionary design paradigm. Fine-grained configurability and capability for rapid reconfiguration allow the balance between, for example, power and fault tolerance to be dynamically "re-optimized" in FPGA-based systems.

C. TOTAL PERFORMANCE METRIC

1. Background

The use of structured methods for evaluating design trade-offs has long been recognized as an important engineering practice. However, as noted in [38] it is extremely difficult to combine performance criteria such as speed, power, simplicity, etc. and produce a "single cost function" for guiding system development. Furthermore, fault tolerance is not typically considered as simply another parameter in this trade-off process. More commonly, in high-reliability systems fault tolerance is considered the preeminent design goal [21]. This line of thinking can preclude solutions that involve modest sacrifices in reliability but yield substantial gains in speed, efficiency, etc. In particular, the impact of high-reliability (achieved with spatial and/or temporal redundancy) on the power consumption of a system historically has not been a major concern.

In recent years, however, there has been more interest is determining and minimizing the cost of fault tolerance. For example, a recent paper by Maheshwari, Burleson and Tessier [29] identifies both fault tolerance and low-power as key design objectives and proposes trade-offs between these two parameters. Their work focuses on VLSI implementation of simple binary counters. They investigate the impacts of including spatial/temporal redundancy, varying transistor dimensions, redesigning circuit elements and several power reduction techniques. However, this work did not involve quantifying fault tolerance and power on a common basis to form an all-encompassing performance metric.

Wang, Ramamritham and Stankovic [81] discuss the idea of a single all-inclusive performance metric. They present a mathematical framework for optimizing the performance of fault-tolerant real-time computing systems. By balancing reliability (gained through task replication) and performance (gained through task completion), an overall "performance index" can be optimized. One of the unique observations from that paper is that a particular computational task has both a "reward" value for being completed and a "penalty" value when it is not completed. By quantifying these reward/penalty values and knowing the expected processor failure rate, one can determine the optimal degree of redundancy that maximizes the performance index. Increasing the redundancy of certain tasks consumes limited resources, processors in Wang's problem, and can degrade the overall performance by inhibiting the completion of other tasks. Though some of the assumptions, goals and constraints in [81] differ from those addressed in this dissertation, the basic philosophy of a quantifiable "performance index" is important in both studies.

Since every measurable performance criteria has some benefit or cost to the overall design, it is important to quantify these values for as many metrics as possible. The following sections and figures explain the general behavior of the most common performance metrics. The graphs in sections 2 and 3 are not intended to represent the relative costs/benefits for each parameter, but to show the behavior of each parameter individually. The relative values of these parameters depend on the particular system being developed and require careful scrutiny of the system's overall objectives. Section 6 presents a detailed example showing how to determine relative weights for each metric.

2. Cost Metrics

Figure 4.1 shows several typical cost metrics for FPGA systems. Although many different cost factors may be applicable to a particular design and application, the metrics described here should be appropriate for most situations. Note that in all cases the costs are monotonically increasing functions of each parameter. The x-axis represents the value of each parameter while the y-axis shows the parameterized costs. The relative importance of each cost parameter is addressed through scaling factors that can be customized for each application. The following paragraphs explain the rationale for the

curves shown in Figure 4.1, based on the scenario of an FPGA-based computing system designed for spacecraft use. This discussion could easily be extended to other technologies and applications.

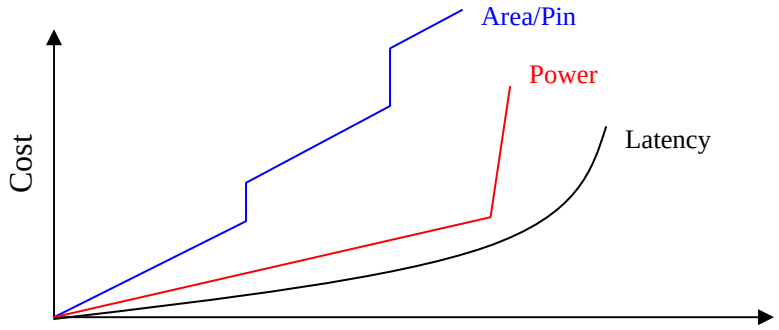


Figure 4.1 Hypothetical Cost Curves

The physical area of a circuit is important because in space applications circuit board area and total volume must be conserved to meet system-level constraints. Circuits that use very little area can be more easily accommodated in the system and allow other functions to share a single board and/or chip. The linear portions of the curve indicate that as a circuit grows it consumes more of the finite resources in the FPGA and prevents other functions from being integrated into the same chip. The discontinuities in the curve occur when a design exceeds the capacity of a certain chip and requires the use of the next larger device in the product family. At each break in the curve, the design moves into a larger and more "costly" FPGA. This may require using a more advanced chip with an equal footprint but higher price, or an entirely different package requiring board-level redesign.

Pin count is often another significant constraint [41] and its cost curve behaves similarly to that of area. The discontinuities in the curve represent jumps when switching between various devices in a particular product line. For example, Xilinx's Virtex XCV600 products (all containing the same internal semiconductor device) are commercially available in 240-, 432-, 560-, 676- and 680-pin packages. Pin count is becoming even more of a concern as semiconductor device technology is shrinking at a faster rate than I/O pin density is increasing. Circuit functionality is generally proportional to chip area, whereas pin count is usually proportional to a chip's perimeter.

Designers have attempted to get around this with unique packaging techniques, such as ball grid arrays (BGAs), although even BGA designs are limited by the physical size of the solder balls and routability from the chip to the package exterior [82]. Despite packaging innovations, I/O pin count continues to be a limiting factor.

Power consumption is an especially significant concern in remote applications such as spacecraft. The cost of using more power typically increases linearly. As a circuit consumes a larger fraction of the spacecraft power budget it prevents other subsystems from receiving the necessary amount of power. At some point the circuit will require more power than is available and the power cost will skyrocket, as shown on the graph as a discontinuity. As an extreme example, the circuit may need more power than the entire spacecraft power system can supply, requiring drastic and costly measures such as increasing the size of the spacecraft solar panels.

Another cost factor is a circuit's latency, or the time required to produce a desired result. A similar metric is throughput, which is the number of calculations completed in a given time period. While throughput is most easily thought of as a benefit, latency is more identifiable as a cost. Although latency and throughput are closely related, it may be instructive to consider them separately. For example, in pipelined systems high throughput may be sustained by high clock speeds even though very long latencies from "deep" pipelining produce results many clock cycles after the inputs are given to the circuit. In non-pipelined systems long computation periods directly affect throughput by limiting the total number of results that can be produced in a finite time period. Whether it is appropriate to account for latency and throughput separately depends on the particular system being considered.

At low latency values this cost is best represented as a linear function, but at longer latencies the cost rises quickly since many systems require rapid data processing in order to work well. This behavior can be best captured by an exponential shape, as shown in Figure 4.1. For example, control systems are often most effective when latency is minimized. A control system can tolerate some latency with relatively little cost or degradation. However, extremely slow computations may require significant redesign of the control system, such as including prediction algorithms to anticipate the system state when control commands are actually issued.

Combining these various costs, the total cost metric in this example is defined as:

$$C_{total} = C_{area+pin} + C_{power} + C_{latency} \tag{4.1}$$

The optimal, but unrealistic, design cost would be zero. Zero cost can be achieved only with a design that requires no area or I/O pins, consumes no power, and produces results instantaneously. In reality, all circuits have some non-zero cost associated with them. Cost metrics other than those described here may be more significant to a particular problem and can augment or replace terms in the preceding equation. In addition, the cost functions for a particular situation may not follow the shapes presented here. Therefore, it is important to identify the appropriate cost metrics for a given situation and make reasonable assumptions about their functional behavior. This information can then be combined with the benefit functions described below to generate a total performance metric.

3. Benefit Metrics

Benefits can also be described in terms of various performance metrics for a given design. Continuing with the scenario of a spacecraft computer, Figure 4.2 shows some of the more common benefit measures. Similar to the cost functions, each benefit term is assumed to be monotonically increasing. The following paragraphs describe each of these in more detail. These benefit metrics are appropriate in most situations and can easily be augmented with other parameters important to a particular design.

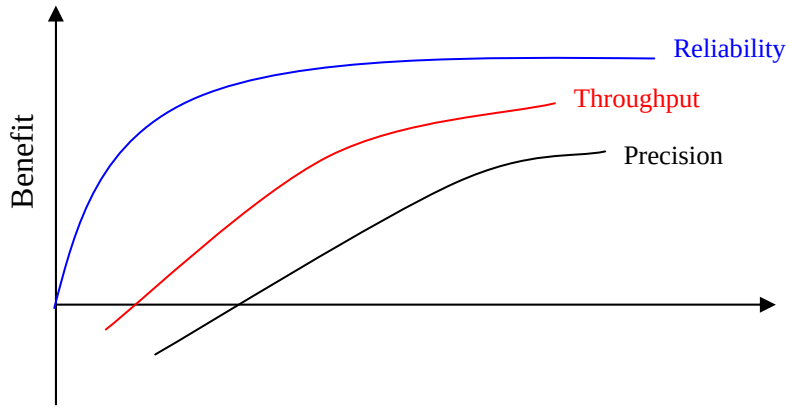


Figure 4.2 Hypothetical Benefit Curves

A critical performance metric in this research is reliability. Reliability can be defined and measured in many ways, such as mean time to failure (MTTF), mean time between error (MTBE), or probability of survival [40]. This dissertation assumes that faults result only from SEUs. A standard assumption is that SEUs are random, uncorrelated events that follow a Poisson distribution [17] due to the interaction of high-energy particles with the semiconductor device. The reliability metric must account for this random process. A key part of estimating reliability is knowing the probability of experiencing a certain number (k) of SEUs in a given time interval (t), as given by [17]:

$$p(k,t|\lambda) = \frac{(\lambda t)^k e^{-\lambda t}}{k!} \quad \text{with } \lambda = \text{mean SEU rate} \tag{4.2}$$

In order to prevent the accumulation of SEU-induced faults in FPGAs, configuration scrubbing must be employed. The Xilinx Virtex FPGA family used in this research is capable of on-line partial reconfiguration, which permits rapid correction of configuration faults. Coupled with fault detection circuitry, this capability ensures that the system returns to its original fault-free condition soon after each SEU. Therefore, MTBE (MTBE = MTTE + MTTR) may be an appropriate metric since it considers both the fundamental failure mechanism via MTTE (mean time to error) and the recovery mechanism via MTTR (mean time to repair) [40]. In most orbital regimes MTTE is much longer than MTTR and therefore either MTBE or MTTE could be used.

As shown in the figure, the reliability term shows a strong logarithmic-like behavior. At lower reliability levels, there is very little benefit from a system that fails often, but the benefit increases sharply for even modest increases in MTBE. At higher levels, the relative benefit of increasing MTBE tapers off considerably. For example, there would probably be little to gain from increasing a spacecraft computer's MTBE from 100 years to 1000 years since even long-lived spacecraft only last 10-15 years.

Since the primary purpose of a computer circuit is to provide output values based on some function of the inputs, an essential metric is the quantity of results produced, or throughput. Implicit in the concept of throughput is the assumption that each solution produced is accurate. For our purposes, accurate means the result is a true representation of the error-free calculation to the circuit's designed level of precision. As shown in the figure, the system derives more benefit when the circuit provides more results. This

relationship is usually linear, though it is possible that the "law of diminishing returns" would be seen beyond a certain point. For example, increasing the refresh cycle on a computer display from 60 Hz to 100 Hz provides essentially no additional benefit because the human observer can't distinguish frame rates above 30 Hz. Another feature to note from the figure is that at the lower performance levels the benefit may actually be negative. This reflects the fact that often there is a minimum acceptable performance threshold. Below this threshold, it would be better to not even use the computer. For example, consider a computerized automobile data display that calculates speed, gas mileage and range-until-empty. If due to some terrible engineering oversight, the range-until-empty calculation was only updated on January 1st of each year, consumers would probably rather not have that feature.

Another key performance criteria is the precision of outputs. With few exceptions, more precise results are better than less precise results. This benefit term also generally has a minimum threshold and an improvement roll-off at higher levels, as shown in the figure above. The shape of this curve could change depending on the units chosen. For example, if precision is measured in terms of real numbers, the function may be mostly linear over most of its range. However, if precision is measured by number of binary digits in the number representation, then the curve may look more exponential.

Combining these various benefits, the total benefit metric is given by:

$$B_{total} = B_{throughput} + B_{precision} + B_{reliability} \tag{4.3}$$

As written above, this function theoretically has no upper bound. However, as explained earlier, all of these terms have an asymptotic upper limit that constrains the total benefit function.

4. Total Performance Metric

Combining the various benefits and costs from the preceding sections, an overall total performance metric (TPM) can be defined as:

$$TPM = Benefit - Cost = \sum_{i}^{N_{benefits}} K_{B_i} B_i - \sum_{j}^{N_{costs}} K_{C_j} C_j \tag{4.4}$$

This general expression allows the inclusion of factors in addition to or instead of those described earlier. In order to relate these diverse parameters, scaling factors *K* are included to reflect the relative importance of each term. The overall performance value is extremely sensitive to these scaling factors, so they must be determined with great care.

Costs and benefits can be evaluated in terms of average values or cumulative values over some finite time interval. The detailed example that follows is based on an average value formulation whereas the approach taken in [81] follows the cumulative value formulation. The cumulative value approach works well in [81] since the authors have a well-defined task set of finite size. The average value approach works better with metrics such as power and throughput that are already defined as average values.

5. Example 1: Generating TPM (Satellite Attitude Control)

Assume that a satellite attitude control system requires a coprocessor that calculates the trigonometric functions sine and cosine to determine commands for maintaining satellite orientation and stability. The top-level design specifications for this hypothetical example are as follows:

precision < 10^{-9}
speed > 1 MHz
power < 1 W
size < ½ of a Virtex XQVR600 FPGA

The design team for this coprocessor has identified the following as the most significant performance factors: speed, precision, reliability, area, latency and power. In order to quantify the trade-offs between various design choices, each performance factor is translated into mathematical expressions as described below.

The physical size of the circuit must be less than half of the predefined FPGA device. Such a specific constraint could be negotiable if the subsystem designers have convincing arguments demonstrating that another device provides better overall performance or that a slight increase in area usage yields improvement in other factors. Nonetheless, the normalized area cost can be expressed as a linear function as discussed in Section 2. By measuring area as a fraction of the total FPGA resources, area cost is given by the expression below. The scaling factor α ensures that the area cost equals one

when the area usage is at the design spec of 1/2. Normalizing costs and benefits allows the relative importance of each criteria to be easily adjusted using the scaling factors *K*.

$$C_A = \alpha A \quad \text{with } \alpha = 2 \tag{4.5}$$

Though not explicitly given as a design specification, a latency requirement can be inferred from the speed requirement. Taking the inverse of 1 MHz and assuming a non-pipelined circuit, one derives a maximum latency of 1 microsecond. However, this derived value may not adequately address the total system needs. Further discussions with the satellite control system design team are necessary to determine whether longer latency may be acceptable, in which case pipelining may be an option. Prior to these discussions, however, a linear latency cost function can be used. With cost normalized to 1 at the derived design spec of 1 μsec, the following function is used:

$$C_L = \lambda L \quad \text{with } \lambda = 10^6 \text{ sec}^{-1} \tag{4.6}$$

The power requirement is fairly straightforward, though this may also be negotiable. Assuming that the circuit's power consumption is reasonable, the power cost function should follow the linear portion of the curve shown in Figure 4.1. As discussed in Chapter III, dynamic power is often directly related to circuit size. Although static power is constant for a given FPGA device, the fraction of static power attributable to a particular function can also be related to that function's size. A linear relationship between area and power is used to simplify the analysis. Normalizing the cost at 1 watt of power consumption leads to the following relationships:

$$\begin{aligned} C_P &= \pi P \quad \text{with } \pi = 1\,\text{W}^{-1} \\ P &= \phi A \quad \text{with } \phi = 2\,\text{W} \\ C_P &= \pi\phi A \end{aligned} \tag{4.7}$$

To evaluate the first benefit factor, speed is seen as synonymous with the throughput parameter from Figure 4.2 above. It is assumed that every result produced by the circuit is correct. The possibility of faulty solutions is subsumed in the reliability factor derived later. Various functions could be used to reflect the key features of a fairly linear behavior at low values and an asymptotic behavior at high values. The formula below is useful because it allows simple manipulation of slopes, "breakpoint" and

asymptote level. This equation is normalized to 1 at a solution rate of 10^6 per second and the asymptote β_S was arbitrarily chosen as 2. For simplicity, the constants are defined by small integers that create a curve reflecting the benefit's anticipated behavior as speed increases. When using this method for a specific application, these constants must be carefully defined based on the particular specifications and behavior of the design.

$$B_S = \beta_S - \eta_S \cdot b^{-\sigma S} \quad \text{with } \beta_S = 2, \eta_S = 3, b = 3, \sigma = 10^{-6} \tag{4.8}$$

The next benefit factor to address is numerical precision. From the design spec of better than 10^{-9} accuracy, the circuit must calculate results with at least 30 bits in the fractional part of each number. Thus the formula for precision is normalized to 1 at the level of 30 bits. Below 30 bits of precision, the value drops quickly as the control system loses the ability to make precise attitude and orientation decisions. However, above 30 bits there is little more to gain since the attitude/orientation sensors and actuators become limitations at such high levels of precision. Thus an asymptote of 1.2 was selected since there is relatively little room for improvement beyond 30 bits. The following equation serves as a starting point for assessing the benefit of high precision. In the equation N is the number of fractional bits in each solution. The exact shape of the curve is sensitive to the exponent base as well as the asymptote and scaling constants.

$$B_N = \beta_N - \eta_N \cdot b^{-\nu N} \quad \text{with } \beta_N = 1.2, \eta_N = 4, b = 20, \nu = 0.033 \tag{4.9}$$

The final parameter to quantify is reliability, which was not specified as an original system requirement. Thus, additional information will be needed for the final system design. In the absence of details, however, a preliminary reliability estimate can be made based on the following assumptions. The only reliability issue considered is radiation-induced logic upsets (i.e., SEUs), though additional issues could easily be included. Reliability is measured by the mean time between error (MTBE) and errors are assumed to occur whenever an SEU affects a non-redundant portion of the circuit. Since the satellite control system operates continuously, the sine/cosine coprocessor needs to operate constantly throughout the satellite's lifetime of roughly 10 years. Finally, it is assumed that most errors cause serious, but not catastrophic, problems. Calculation errors in the sine/cosine coprocessor can necessitate a time-consuming reorientation of

the spacecraft and subsequent mission interruption, but they are not expected to lead to total loss of the spacecraft.

These assumptions are used to formulate an equation describing the benefit of reliability. At low reliability levels the system will require frequent reorientation maneuvers, preventing the spacecraft from conducting much of its mission. High reliability is obviously good, though extremely high reliability may not be cost effective compared to more modest reliability levels. Over a 10-year mission lifetime it is reasonable to assume that a small number of recoverable errors, for example one per year, is acceptable. Thus the reliability benefit function given below follows an inverse exponential curve and is normalized to 1 for a MTBE of one year. With adequate fault masking, it is reasonable to expect MTBE values in the range of months or longer in LEO environments and shorter MTBE in higher radiation environments.

$$B_R = \beta_R - \eta_R \cdot b^{-\rho R} \quad \text{with } \beta_R = 1.2, \eta_R = 1.2, b = 6, \rho = 1\,\text{year}^{-1} \qquad (4.10)$$

Combining all 6 benefit and cost factors explored in this hypothetical example, TPM is given by:

$$TPM = K_{B_S} B_S + K_{B_N} B_N + K_{B_R} B_R - K_{C_A} C_A - K_{C_L} C_L - K_{C_P} C_P \qquad (4.11)$$

Note that TPM is a unitless quantity since the benefit and cost terms are all normalized quantities.

The final step in generating the total performance metric is to establish the relative importance of each factor by determining the scaling factors *K*. A starting point might be to assume that all factors are equally important, in which case each *K* can be set to 1. More realistically, some factors are more important than others. For illustrative purposes, Table 4.1 presents scaling factors for three different prioritization schemes. Scheme A assumes all factors are equally important, Scheme B emphasizes reliability, and Scheme C emphasizes precision.

Factor	Prioritization Scheme		
	A	B	C
Speed (S)	1	1	0.5
Precision (N)	1	1	2
Reliability (R)	1	2	1
Area (A)	1	0.5	1
Latency (L)	1	1	0.5
Power (P)	1	0.5	1

Table 4.1 Possible Cost/Benefit Prioritization Schemes

For illustrative purposes, the figure below shows the behavior of each parameter using the scaling factors from prioritization scheme B. The x-axis labels represent various levels of each parameter relative to the design specification or derived requirements discussed earlier. For example, "1x Spec" corresponds to:

speed = 1 MHz
precision = 10^{-9}
reliability = 1 year MTBE
size = ½ of a Virtex XQVR600 FPGA
latency = 1 μsec
power = 1 W

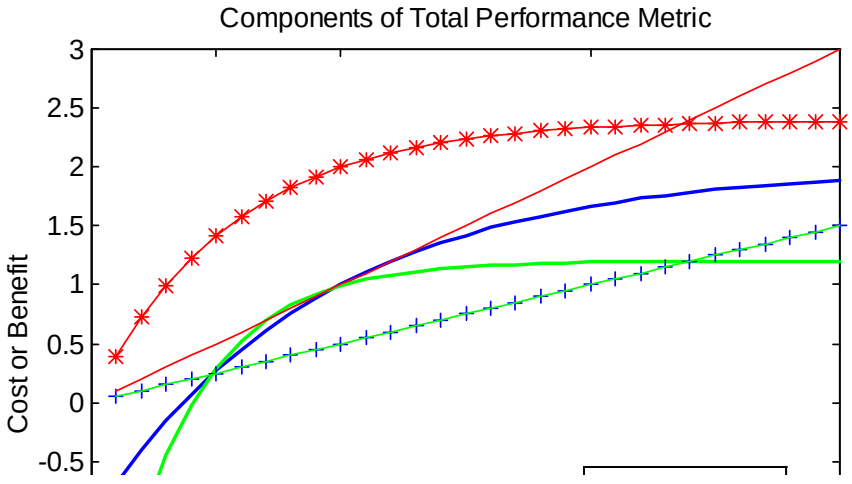


Figure 4.3 Hypothetical Functional Behavior of TPM Components

Though interesting, this figure cannot be used directly to make trade-offs between the various design parameters. One cannot simply move along each curve independently since the parameters are strongly related to one another. In this example, area and power have a simple linear relationships. However, the interactions between power-reliability or precision-latency, for example, are difficult to anticipate accurately. With a practically infinite design solution space, an exhaustive analysis is impractical. Instead, it is more reasonable to investigate a small number of points that span the design space. The results from these early predictions can help focus and refine the more detailed design and analysis work.

To conclude this example, the TPM is calculated for several plausible design solutions using each of the prioritization schemes listed in Table 4.1. Based on some of the assumptions used in this example, the design space can be divided into four factors. Table 4.2 below lists the factors used in this analysis and the TPM values calculated from the equations derived above. Since area and power are directly related in this example, a single "size" factor can be used, with "large" meaning a circuit twice the specification value and "small" meaning a circuit half the specification value. For simplicity, latency and throughput can likewise be combined into a "speed" factor. Thus "fast" corresponds to latency of half specification and throughput of twice specification, and "slow" means these terms are swapped. Precision and reliability cannot be easily combined with other factors, so they are treated separately. For these two factors, "high" represents twice the spec and "low" means half the spec. For added realism, the shaded regions in the table represent design points that the engineering team predicts cannot be achieved. Thus these TPM values are unattainable. From the remaining design options, the best designs for each prioritization scheme are highlighted.

			Low Reliab.			High Reliab.		
			A	**B**	**C**	**A**	**B**	**C**
Low Precision	Small	Slow	-1.7	-0.5	-0.6	-1.3	0.4	-0.1
		Fast	**1.2**	2.4	0.9	1.6	3.3	1.3
	Large	Slow	-4.7	-2.0	-3.6	-4.3	-1.1	-3.1
		Fast	-1.8	0.9	-2.1	-1.4	1.8	-1.7
High Precision	Small	Slow	-0.8	0.4	**1.2**	-0.4	1.3	1.7
		Fast	2.1	3.3	2.7	2.5	4.2	3.1
	Large	Slow	-3.8	-1.1	-1.8	-3.4	-0.2	-1.3
		Fast	-0.9	1.8	-0.3	-0.5	**2.7**	0.1

Table 4.2 Hypothetical TPM for Various Design Points

6. Example 2: Determining K Factors (Satellite Image Processing)

As highlighted in the previous example, the scaling factors *K* heavily influence the TPM. Different *K* weighting schemes lead to different conclusions about which solution is optimal. Therefore it is important to determine them carefully based on the context of the particular design problem. The *K* factors serve two primary roles. First, they influence the slope of the TPM as a function of each parameter around the nominal design point. Secondly, they establish the relative value of the cost/benefit parameters at extreme high/low values. The following example describes in more detail the types of considerations that help establish these *K* factors. Keep in mind that the *K* values could differ greatly for a given design depending on the application.

In this hypothetical example, the task is to build an image compression processor for a commercial satellite imaging system in order to reduce the data downlink bandwidth requirements. The satellite's main function is to collect visible and infrared images of specific ground locations for customers such as governments, regulatory agencies, farmers and private landowners. The business model for such a product has been well established by satellite operators such as Space Imaging (IKONOS) and DigitalGlobe (QuickBird). In this scenario there is a strong financial motivation for developing cost effective solutions. Therefore it is worth considering options that may sacrifice some degree of reliability or precision in order to reduce the costs involved. On the other hand,

it is important to sustain the collection, processing and delivery of imagery products. The company can only make money as long as it continues to provide images to customers.

The same performance factors used in the previous example are also applicable to this design. Similarly, the reliability efforts here focus on tolerating radiation-induced SEUs. A plausible prioritization scheme for the TPM factors is listed in Table 4.3. The following paragraphs explain the reasons for assigning these particular *K* values.

Factor	Scaling Factor (K)
Throughput	40
Reliability	10
Precision	8
Power	2
Area	2
Latency	1

Table 4.3 Example Cost/Benefit Relative Weighting Factors

The following discussion focuses on the behavior of each parameter around the nominal design solution, called the "1x Spec" point in the previous example. The baseline specification is given as:

throughput = 1 Hz
reliability = 90%
precision = 16 bits
area = 1 Virtex XQVR600 FPGA
power = 5 W
latency = 1 sec

For simplicity, assume that all parameters behave linearly and with equal slopes near this point. Thus only the *K* factors alter the slope of each cost or benefit term. For example, the *K* values determine whether or not a 10% improvement in precision is more beneficial than a 10% improvement in area.

Throughput is the most important design criteria. As mentioned earlier, the mission of the spacecraft is to produce images, thus the ability to maintain a high data rate is essential for maximizing revenue. A large *K* value for this factor ensures that the design does not sacrifice throughput. The value 40 that appears in the table, though

arbitrary, was selected to accommodate integer values for the remaining *K* factors. There is no requirement that integers be selected for these scaling factors, but they are used here for simplicity.

Reliability is the second most important parameter since low reliability can lead to several negative consequences. First, it degrades the satellite's ability to collect and transmit images on demand and therefore reduces the effective data throughput. If failures coincide with actual imaging operations, valuable image data will be lost. Additional data collections may need to be scheduled for subsequent orbits in order to satisfy customer demands, which in turn will limit the number of customer requests that can be satisfied. Furthermore, customers may become displeased and turn to other service providers.

In this hypothetical system, a 90% reliability is required by the baseline system specification. Thus for every 100 images collected, 90 of them should be properly processed and transmitted. The following figure shows what happens to the data output rate when the processing throughput and reliability are individually reduced to one-half the specification level. As can be seen, reducing throughput by half while reliability stays at 90% causes the data rate to drop to 45 (a decrease of 45 images). By comparison, a reliability reduction of half while throughput remains at 100 decreases the data rate to 80 (a decrease of 10 images in the same time interval). The slope for throughput should be steeper than for reliability since its impact was much greater (reducing the data image rate by 45 versus 10). Therefore for this application, reliability is assessed to be roughly one-quarter as important as throughput, and is assigned the *K* value 10.

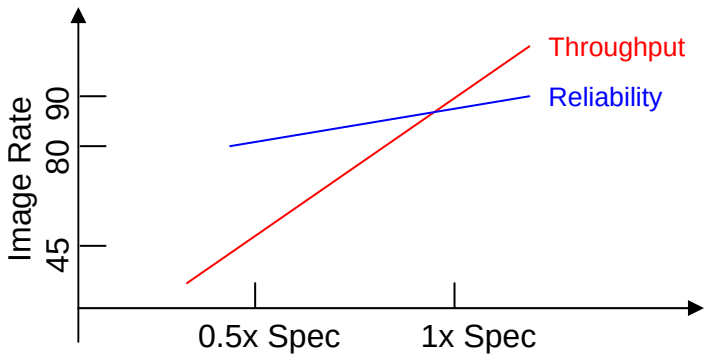


Figure 4.4 Hypothetical Benefit Curves

Precision is an important TPM component in this example since it determines the quality, and therefore value, of the images produced. Considering that different customers have different image quality requirements, a price scale is established with reduced prices for lower quality images and premium prices for the highest quality images. At the baseline precision level of 16 bits, the price is $5000 per image. A discounted price of $4500 per image is offered for 8-bit resolution. This price reduction of $500 can be considered lost revenue. An equivalent reduction in revenue would occur if throughput were decreased by 10%, from 1 Hz to 0.9 Hz. To properly weight the precision term, a *K* factor of 8 is required.

Power and area are less important for this application than they typically are in satellite systems. The amount of power and area needed for this image compression processor are small fractions of the overall satellite resources. On this hypothetical satellite the solar panels generate an average of 500 watts, thus even large increases in this processor's consumption require only modest upgrades. A doubling of the processor's power usage from 5 W to 10 W requires a 1% increase in solar panel capacity. Putting this into financial terms, assume that the solar panel subsystem for the spacecraft costs $1M and that a 1% increase in output can be achieved for an additional $10,000. For comparison, an advanced semiconductor processing technology that improves the SEU immunity of the device can be purchased for $10K. This new device will reduce the probability of error from the baseline design level of 0.10 to 0.08, a 20% improvement. Thus, in this simple example a 100% increase in power and a 20% increase in reliability have equal monetary value. Since reliability has a *K* value of 10, power should be assigned the value 2 to ensure proper scaling between these terms.

Similarly, since this processor is allocated a single FPGA chip that occupies a small percentage of the total satellite volume, modest increases in chip area are possible. If the design requires more circuitry than the baseline FPGA provides, it is also possible to switch to a newer, more dense device since the cost of FPGAs is small compared to the total program cost. Putting chip area into financial terms, assume that upgrading from the baseline FPGA device to one with twice the logic capacity costs $10K. Following the rationale for the power *K* factor, the *K* value for area is also 2.

Latency is the least important factor since customers for this type of satellite imagery typically have no demand for real-time data. Although zero latency is ideal, reasonable amounts of latency cause little degradation. In this application, the main forces limiting latency are on-board data storage and ground station access periods during which data is downlinked. Some amount of data buffering can be included in the design, but a large amount of data storage on the satellite is undesirable. A nominal latency value of 1 second is specified to match the data collection rate of 1 Hz.

For this example, assume that the satellite operations procedure includes the collection of a second image of each ground location, if necessary. If the first image is corrupted by clouds, sensor noise or smearing from satellite vibration, the second image is collected. The decision whether or not to collect this second image is based on an automatic quality check on the fully-processed first image. This quality check must be made within 40 seconds to ensure the satellite does not pass beyond view of the target. A latency of 40 seconds or longer means that the second image must be collected on every target, which is equivalent to reducing the throughput by half. Thus a K value of 1 provides the proper weighting of this parameter.

7. Accounting for Reliability Factors

One potential problem with the parameter set described in the preceding sections is improper accounting of reliability. Reliability is treated as a separate benefit factor, although it can influence both the "throughput" and "precision" factors. If errors are infrequent it is acceptable to consider reliability separately and ignore reliability when estimating these other factors. As errors become more common, however, it may be more appropriate to include reliability in the throughput and/or precision factors.

It is easy to see that errors decrease the number of correct solutions produced by the circuit. Thus, instead of having a separate factor, reliability could be accounted for by adjusting the expected number of correct solutions. Reliability also influences the average numerical precision. If an RPR implementation is chosen to achieve fault tolerance, some SEUs will not lead to failure but will cause the circuit to produce solutions with less than the full precision. If this happens rarely, the system receives nearly the same quality of service as with a more comprehensive redundancy structure,

such as TMR. However, if this happens relatively often, the quality of service is significantly degraded. One could account for this quality factor by measuring the precision of each solution. The variable *N* in Equation 4.9 could represent the total number of fractional bits (i.e., precision) for all correct solutions produced in a finite time period.

SEUs in space are fairly rare in most orbital regimes. The average time interval between SEUs range from minutes to hours [17] depending on altitude as well as solar activity and geomagnetic conditions. In geosynchronous orbit, Bridgford predicts that current FPGA devices such as the Xilinx Virtex-II will be affected by 1 to 30 SEUs per hour [83]. Furthermore, such devices may experience SEFI-type events (i.e., SEUs that lead to errors requiring complete device reset and/or power cycling) only once in 60 years of continuous operation [83]. It would be meaningless to apply such low error rates over short timescales. In a one second period there is an extremely low probability of experiencing an SEU-induced error. Assuming a Poisson distribution of SEU arrivals [17] and an average SEU rate of 1 per hour, the probability of experiencing at least one SEU in a one second period is only 0.000278. Thus the expected throughput would be reduced from 1,000,000 to 999,722 calculations per second and precision reduced from 30 to 29.99 bits. Such small numerical effects do not adequately capture the importance of high reliability in this situation.

Properly accounting for reliability in the total performance metric is challenging and can be approached in various ways. Assigning a benefit value to various levels of reliability, as done above, is appropriate for most situations. Determining a reliability benefit value is impractical for certain applications, so alternative methods should be considered. One approach is to create a family of cost-benefit isocurves for various levels of reliability as shown in Figure 4.5. Each isocurve plots the lowest cost (x-coordinate) design that achieves the corresponding benefit (y-coordinate), where the total benefit value is summed over the remaining benefit terms once reliability is isolated. The system designer can examine this family of curves to find efficient design options that meet the desired reliability level, similar to goals stated in [43]. The knees in the curves in this figure are likely to be the most efficient design points.

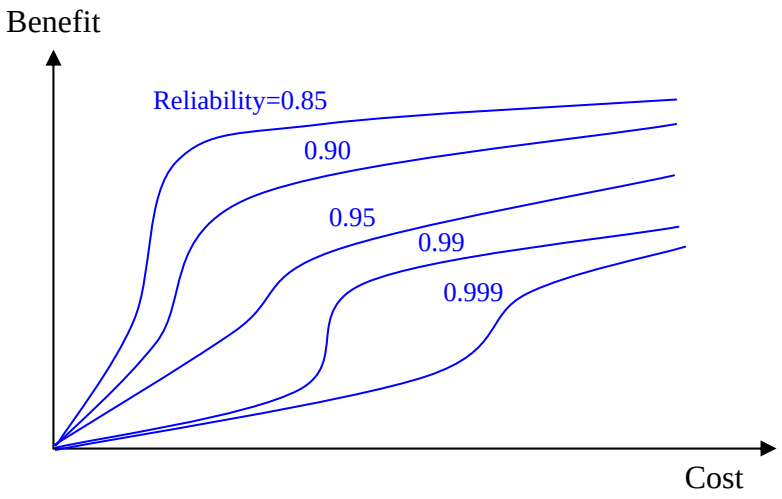


Figure 4.5 Conceptual Reliability Isocurves within Cost-Benefit Design Space

D. MEASURING RELIABILITY

Whereas it is relatively straightforward to measure performance factors such as speed and power, determining the reliability or fault tolerance of a design can be challenging. The first step is deciding which kinds of faults are expected. Should the designer be concerned with only transient faults such as SEUs, or also permanent faults due to device fatigue and burnout? This dissertation focuses on transient faults in an FPGA's configuration memory caused by SEUs. However, in critical applications such as life-support systems, all types of faults and failures should be considered [40].

Once the class of faults has been defined, the next step is to estimate and/or measure the system's response to each possible fault. This step can be very complicated considering the size and complexity of modern electronic systems. For example, in the Xilinx Virtex XQVR600 device there are over 3.3 million configuration bits that can be upset by SEUs. If one considers the possibility of multiple simultaneous SEU-induced faults, the problem quickly becomes intractable. For example, there are over $5*10^{12}$ possible two-SEU combinations in the Virtex device! There is no reasonable way to exhaustively simulate or test each of these combinations. Virtually all FPGA fault tolerance studies consider only individual SEU effects. To date, the only thorough work with multiple SEUs in FPGAs has mainly involved SEUs affecting physically adjacent bits on the device [84]. However, the preponderance of SEUs affect only a single bit. Therefore the fault model assumed in this dissertation is a single SEU-induced bit flip in

the FPGA configuration memory. All possible single bit upsets should be thoroughly analyzed.

The system's response to each fault must be carefully monitored and categorized as either error-producing or non-error-producing. Many researchers use the terms "sensitive" and "non-sensitive" to characterize particular configuration bits in an FPGA design [4], [42]. A sensitive bit is an FPGA configuration bit that causes errors in output data when corrupted by an SEU. Certain data errors may only be manifested when particular input data patterns are applied to the circuit [4]. Therefore some faults may appear to be non-error-producing if testing involves only a small selection of possible inputs. With large input vectors in some circuits, such as a 32-bit multiplier, it would be extremely time consuming to test every possible input data combination. Thus, a compromise must be reached between testing a small number of inputs and testing every possible input combination.

Sensitive bits can be further categorized as either "persistent" or "non-persistent" according to whether their impact persists after a configuration scrub [5], [8], [42], [85]. A persistent error, such as corruption of the contents in a finite-state machine, requires rewriting the configuration memory and resetting the system. Therefore SEU sensitivity experiments must be carefully constructed to identify and correct persistent errors as they occur. Otherwise the long-term effects of persistent errors make it difficult to evaluate subsequent faults. After analyzing all configuration bits for a particular FPGA design, the overall configuration sensitivity can be calculated as the percentage or total count of sensitive configuration bits. The fault tolerance of various designs can compared based on this overall sensitivity. Those with lower configuration sensitivity are likely to be more SEU-tolerant.

Another challenge is deciding whether a particular fault/error leads to operational failure or a lesser degree of system degradation. For example, in an RPR implementation many faults are likely to degrade the precision of output data. Should this situation be considered a true operational error? Perhaps an isolated occurrence of imprecise outputs is acceptable, but frequent and/or numerous consecutive events might be unacceptable.

The final step is to estimate the system's reliability by predicting its response in a realistic environment. This calculation is based on anticipated fault rates (due to factors

such as radiation levels) and the probability that faults in the device will lead to error conditions. High reliability can be achieved if the radiation conditions are benign or the device has low susceptibility to error-producing faults. Conversely, low reliability occurs when the radiation environment is high and the device is likely to produce errors from many fault conditions.

1. Assumptions

The intent in this dissertation is to compare the relative reliability of various designs rather than calculating absolute reliability or fault tolerance values. Therefore SEU sensitivity, based on the number of configuration bits capable of causing errors, is an adequate metric for comparing the reliability of different FPGA designs. The following two sections provide additional background information on several aspects of determining absolute SEU rates in space.

This analysis does not address the issue of varying SEU susceptibility of different FPGA structures. As studied by Ceschia [44], the energy spectrum of radiation particles influences the statistics of SEUs on an FPGA. For example, Ceschia found that LUT configuration bits on the Xilinx Virtex XCV300 device were the most sensitive and could be upset by relatively low-energy particles. Therefore, a more accurate reliability estimate can be achieved by considering the radiation spectrum in the operating environment. Data from [44] shows roughly a factor of 2 difference between the most sensitive and least sensitive FPGA elements.

Nonetheless, sufficiently accurate results can be achieved by assuming equal sensitivity for all FPGA configuration bits. Most work in the literature, such as [86], does not consider energy-dependent bit susceptibility since there are larger sources of uncertainty in reliability predictions. For example, the actual space radiation environment is not precisely known and varies significantly in time. Furthermore, the complex interaction within FPGA circuits causes considerable variability in measured bit sensitivity between different design configurations [6]. For purposes of comparing the reliability of competing designs it is usually safe to assume a uniform SEU susceptibility. Most designs with similar functionality use similar proportions of LUT, MUX, routing,

etc. elements [1]. Thus, the relative reliability of various design implementations can be assessed fairly by assuming all configuration bits are equally susceptible to radiation-induced upset.

2. SEU Rates in Space Environment

The space radiation environment consists mostly of ions and protons of varying energy levels. As discussed in [17], different equations govern the heavy ion response and proton response of microelectronic circuits. Fortunately, most satellite orbital regimes are dominated by one radiation source or the other. Thus predictions of on-orbit performance can focus on the radiation source appropriate for the spacecraft orbit. At geosynchronous altitude (36,000 km) the predominant effect is due to cosmic rays, which are highly energetic ions of solar and extra-solar origin. At lower altitudes, in particular within the lower Van Allen belt which peaks at around 10,000 km [17], [87], magnetospherically-trapped protons dominate. Some typical formulas used for ion and proton SEU rates as presented in [17] are given below:

$$f_{ion} = 22.5\pi\sigma_a Q_c \int_{22.5Q_c/s_{\max}}^{L_{\max}} \Phi(L)c(s(L))dL/L^2 \tag{4.12}$$

$$f_{proton}(A) = \int_A^\infty \sigma_p(A,E)\Phi_p(E)dE \tag{4.13}$$

SEU rate depends strongly on the device's geometry, composition and other physical parameters. Parameters such as Q_c (critical charge) and σ_p (proton SEU cross section) can be estimated analytically or, more often, measured experimentally. Smaller circuit dimensions and reduced transistor switching energy in modern microelectronics make new devices much more susceptible to direct proton-induced SEU. Messenger and Ash [17] estimate that as device feature sizes drop below 0.3 μm, this direct SEU phenomenon will increase dramatically. Figure 4.6 shows estimated SEU rates for a notional 64 Mbit memory module in a particular orbit as the critical charge, which is a strong function of circuit dimensions, is varied.

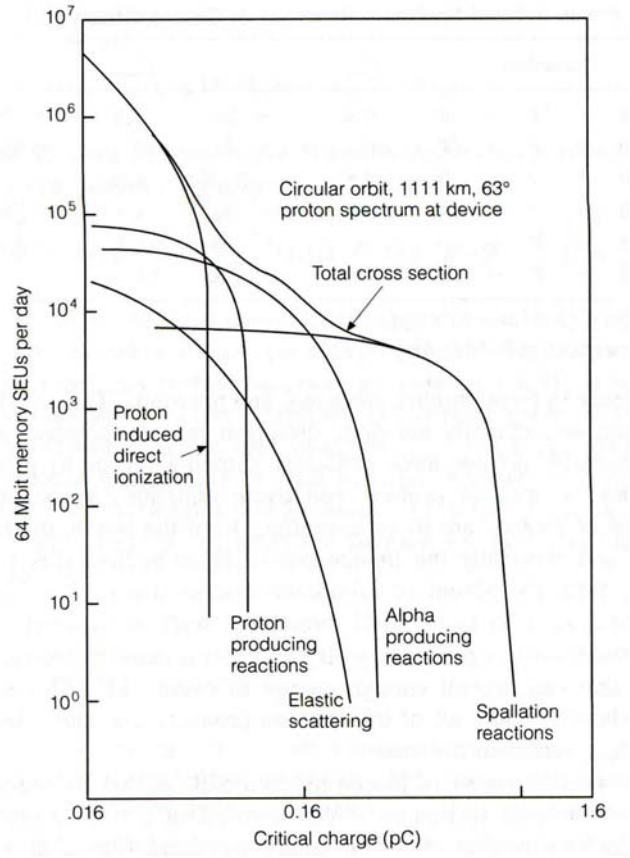


Figure 4.6 SEU Rates for Notional 64 Mbit Memory (from [17])

Though the equations and phenomenology governing ion-induced and proton-induced SEUs differ somewhat, upset rates due to both sources can be simplified to the following equation:

$$f_{SEU} = \sigma_{SEU} \cdot \Phi_{rad} \qquad [\frac{\text{cm}^2\text{events}}{\text{particles}} \times \frac{\text{particles}}{\text{cm}^2\text{s}}] \tag{4.14}$$

Despite the simplicity of the equation, determining values for cross section and flux involves considerable effort. Cross section must be determined for each constituent of the radiation environment or averaged over all particles of interest. Flux can be measured in various ways, but is typically calculated as the number of particles within a specified energy level that pass through a given area or volume in a certain time period. Extensive experimentation and analytical modeling have been conducted to characterize the space radiation environment. In fact, the first US satellite, Explorer 1, was launched

in 1958 with instrumentation developed by Dr James Van Allen that led to the discovery of the radiation belts. Various computer models, such as CREME (Cosmic Ray Effects on Microelectronics) and others available within SPENVIS (Space Environment Information System), are used extensively in current radiation effects studies. The actual flux incident on spacecraft circuits is also influenced by spacecraft shielding effects. Thus, environmental flux values must be adjusted using scaling factors that account for these shielding effects.

Flux varies according to orbital altitude, inclination, and solar activity. At higher altitudes the proton population is negligible and cosmic ray sources dominate. Radiation levels may increase by several orders of magnitude following extremely strong solar flares. A worst-case GEO environment during an "anomalously large solar flare" could generate 7,740 SEUs per day in the Xilinx Virtex-II XQR2V6000 device [86]. Normal GEO conditions would be about 1/2000 this amount [17], or about 4 per day. Estimates for a Virtex XQV1000 device in a 560 km LEO orbit range from 0.13 to 4.2 SEUs per hour depending on solar activity and geographic location [1].

Rough estimates of SEU rates are given in Figure 4.7, which highlights the preponderance of proton-induced SEUs within the lower Van Allen belt. The data for Figure 4.7 was produced in the late-1980's and estimates orbit-dependent SEU rates of between 10^{-6} and $5*10^{-5}$ upsets/bit/day [17]. Tiwari uses an experimentally-derived value of $48*10^{-6}$ upsets/bit/day [80] based on proton radiation tests on a Virtex XQVR300 device. Somewhat surprisingly, this number is within the range of values from [17] that were generated about 15 years earlier.

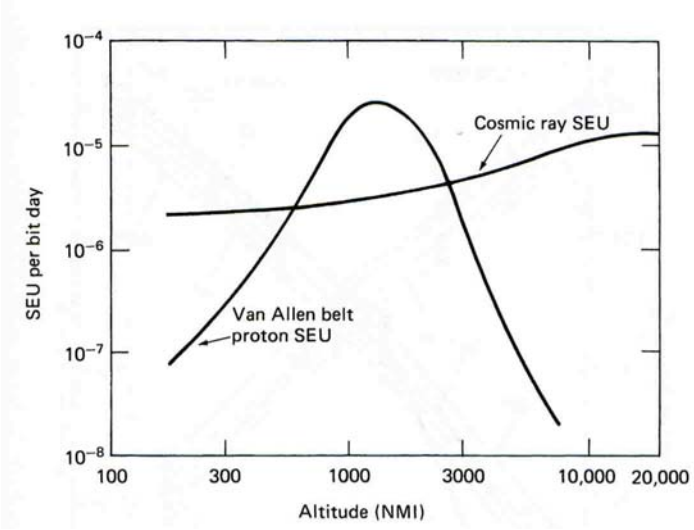


Figure 4.7 Typical Altitude Variation of SEU Rates for 60° Orbit (from [17])

Although actual SEU rates depend on the particular integrated circuit under study, estimates can be made using data from similar devices. For example, Messenger suggests that if data is not available for the device under investigation, one can use proton susceptibility data from "part types that are alike or similar in … technology or … function." [17]

3. SEU Cross Section

An important measure of SEU sensitivity is the cross section. Cross section represents the theoretical area within which a transiting radiation particle with sufficient energy would cause an SEU. Thus a large cross section is undesirable. Cross section can be calculated for individual bits, for various resource classes as in [44], or for entire designs as in [6]. SEU cross sections can be derived experimentally using the following equation from [17]:

$$\sigma_{SEU-particle} = \lim_{t\to\infty}\left(\frac{N_{SEU}}{\Phi_{particle}}\right) \quad [\text{events/s} \div \text{particles/cm}^2\text{s}] \qquad (4.15)$$

Cross sections for various FPGA devices have been calculated and published extensively in the literature. For example, experiments on the Virtex 300 devices showed a proton saturation cross section of $2.2*10^{-14}$ cm^2 [88] and ion saturation cross section of

between $2*10^{-8}$ and $8*10^{-8}$ cm^2 [44], [89] per configuration bit. When discussing cross section values, it is important to understand the distinction between static and dynamic cross section. As explained in [85], static cross section is measured without a clocking signal running and is therefore independent of the design instantiated on the FPGA device. Dynamic cross section is measured with a clocking signal driving the particular design programmed onto the device and depends on how the design uses various FPGA resources. Dynamic cross section can be calculated by multiplying the static cross section by the ratio of sensitive configuration bits to total configuration bits.

4. Reliability and Mean Time Between Error

MTBE, the reliability measurement needed for the total performance metric, can be estimated from cross section and SEU rate data. Equation 4.16 shows that MTBE is simply the product of the SEU rate and the sensitive cross section fraction. The SEU rate is determined by the orbit parameters and the physical behavior of the FPGA device. It may be estimated directly from data such as that presented in Figure 4.7 or determined from the radiation flux and circuit dynamic cross section. Since the dynamic cross section is design-dependent, it is the key parameter for comparing the reliability of different designs. Depending on what data is available, one of the equivalent formulas below can be used.

$$MTBE = f_{SEU} \cdot \sigma_{fraction} = f_{SEU} \frac{\sigma_{dynamic}}{\sigma_{static}} = \Phi_{rad} \cdot \sigma_{dynamic} \tag{4.16}$$

It should be noted that configuration scrubbing is required to ensure that this equation is valid. The preceding MTBE calculation assumes that the FPGA circuit is reconfigured soon after the onset of an error-producing SEU. Configuration scrubbing is essential in FPGA design to prevent the accumulation of SEU-induced faults [85]. Scrubbing can be scheduled periodically or can be performed in response to detected faults/errors [42], [85]. In either case, the scrub rate should be considerably faster than the SEU rate to minimize the possibility of multiple SEUs affecting the circuit simultaneously. Nearly all current efforts in FPGA fault tolerance incorporate some form of configuration scrubbing [6], [8], [49], [80], [85].

An additional class of failures that must be considered is known as Single Event Functional Interrupt (SEFI). SEFI refers to SEU-induced faults that cause especially disruptive and persistent problems. Often, SEFIs can only be corrected through complete power cycling of the device. This special class of failures has been attributed to unique components within modern FPGAs. For example, Graham [2] identifies three possible sources for SEFIs on Virtex devices: 1) JTAG controller, 2) power-on-reset state machine and 3) SelectMAP configuration pins. SEFI events are much less common than normal SEUs since they can only be caused by a small number of bits on the device. However, there are no mitigation methods for some SEFIs. Thus the probability of SEFI occurrence sets an upper limit to the maximum achievable reliability of an FPGA design. One method of accounting for SEFI is to augment the regular configuration cross section value with an additional factor, as shown in the following equation [86]. The total SEFI cross section for Virtex devices is estimated at about 10^{-5} cm^2 [2], [86].

$$\sigma_{total} = \sigma_{SEU} + \sigma_{SEFI} \tag{4.17}$$

5. Reliability Comparison Between RPR and TMR

Ideally, the methodology developed above could be used to assess the effectiveness of Reduced Precision Redundancy (RPR) versus TMR in improving reliability for FPGA systems. Unfortunately, such a comparison cannot be made in a general sense. Since dynamic cross section is design-dependent, one must choose specific designs to analyze. Chapter VI presents the analysis from several test designs. However, some general observations can be made about RPR versus TMR. In addition, when analyzing RPR reliability it is crucial to account for instances when the circuit provides less precise, but nonetheless "correct" results. This section discusses these general issues.

TMR failure can occur only when 1) there are simultaneous faults in 2 or 3 of the redundant modules or 2) there is a fault in the voter circuit or related control/output logic. The reduced precision architecture is also sensitive to simultaneous faults in multiple modules and SEUs in the voter/control circuitry. The main drawback of RPR is that precision is lost whenever an SEU causes an error in the exact solution. When the exact result disagrees with the higher-confidence approximate results, the most probable

scenario is that a fault exists in the exact module. Since the exact module is much larger, it is more likely that a random SEU occurred within the exact module than in another part of the circuit. Therefore the voter should pass forward one of the lower precision results. In contrast, a TMR design can handle single errors in any computation module without loss of precision.

Although the more extensive redundancy of TMR might seem more reliable than the reduced precision approach, there are some subtleties worth investigating. For example, assume a circuit for which the exact solution occupies 1/3 of the FPGA, and voter circuitry that is negligible in size. This second assumption will not always be valid but simplifies the following discussion. Compare this to an RPR design that uses an exact module occupying 1/3 of the chip area, two approximate modules that each occupy 1/30 of the chip, and a voter that is negligible. Figure 4.8 depicts the hypothetical FPGA layouts for these designs. The most obvious difference is that the RPR design occupies much less chip area, making it less susceptible to permanent device failures affecting small localized regions. Another advantage of RPR is that the small approximate modules can be enhanced with additional redundant modules and distributed voting units to minimize the unmitigated cross section.

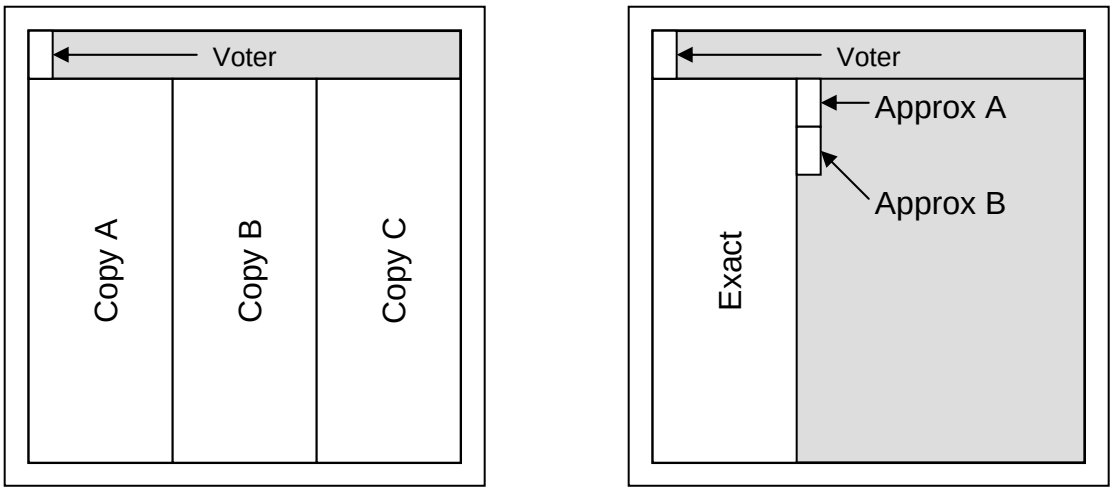


Figure 4.8 Hypothetical TMR circuit (left) and RPR circuit (right)

At low SEU rates, spatial redundancy with configuration scrubbing can allow a system to continue functioning while faults are corrected in the background. At higher SEU rates the accumulation of errors can occur faster than they can be corrected. The TMR design in Figure 4.8 is protected against most single faults. The only single fault susceptibility is in the voter/control/output logic. To protect against multiple faults, the configuration scrub cycle (the time for a fault to be detected and corrected) must be less than the time it takes for 2 modules to suffer SEUs. Though improbable, it is not impossible for successive faults to occur within a relatively short scrub cycle. Following a first-fault event, roughly 2/3 of the chip is vulnerable to successive faults until the configuration scrub corrects it.

In contrast, the RPR design is less vulnerable to consecutive SEUs for two reasons. First, it requires much less time to scrub the approximate modules, so an error in that level can be corrected quickly before another SEU hits. Second, since it occupies less physical area it has a smaller sensitive area (i.e., a smaller SEU cross section) and is therefore unaffected by SEUs occurring over much of the chip. In this hypothetical design, about half of the FPGA is unused (or used for other purposes) so roughly half of all SEUs will not affect the computation. Of the SEUs that occur in the functional circuit, about 90% will occur in the precise module. For the few SEUs that occur in the approximate modules, an intelligent voter and redundancy within/between approximate modules protect against data or configuration errors.

Finally, the reliability factor must account for the less precise outputs produced by the RPR configuration. Though the reduced precision values provide less "benefit" than full precision results, they nonetheless prevent the system from experiencing a failure as it would if it received incorrect data. An intermediate "benefit" value should be attributed to these less precise solutions. Though the probability of error with TMR might be very low, such errors often deviate greatly from the correct result. Building upon the cost/benefit concepts presented earlier in this chapter, one might assign TMR answers a value of B_{max} when correct or 0 when incorrect. However, with RPR it is reasonable to define three benefit levels $\{0, B_{mid}, B_{max}\}$. The reliability score can then be scaled according to the anticipated frequency of precise, imprecise, and incorrect solutions:

$$B_{reliability-TMR} = p_{correct} \cdot B_{max}$$
$$B_{reliability-RPR} = p_{correct} \cdot B_{max} + p_{imprecise} \cdot B_{mid} \quad (4.18)$$

As an example, consider a TMR system that provides correct answers 95% of the time, and an RPR system with 8% imprecise and 2% failure probabilities. Assuming a B_{max} value of 1 and B_{mid} value of ½, the reliability score for TMR is 0.95 compared to 0.94 for RPR. Thus in this example TMR has a higher reliability score, but only by a small amount.

These simple models do not address the possibility of failure due to the cumulative effect of multiple successive imprecise results. For example, the hypothetical satellite control system from earlier may be robust enough to accept imprecise solutions 1% of the time if they are distributed over time, but may drift outside a safe tolerance range if many occur in rapid succession. Accounting for this effect would add considerable complexity to the analysis. Since SEUs occur relatively infrequently and configuration scrubbing is used in nearly all designs, most situations do not require this extra complication.

E. SUMMARY

This chapter introduced a formal method for comparing the overall value of competing design solutions using a Total Performance Metric (TPM). By considering all relevant performance factors simultaneously, the TPM helps engineers objectively select the most advantageous designs. Compared to conventional system development processes that often rely on subjective evaluations, this quantitative method ensures fair and consistent decisions. Applying the TPM approach to real-world problems can reveal interesting trade-offs. For example, RPR designs that provide lower reliability and/or precision than TMR may be more beneficial overall if they offer significant advantages in power, speed, etc.

The following chapters illustrate the potential benefits of RPR by quantifying several TPM parameters for a test circuit based on the CORDIC algorithm. Chapter V describes the CORDIC algorithm and explains why it was chosen as the test case in this research. Chapters VI and VII show that, although TMR has better overall reliability,

RPR is effective for improving reliability of the most significant bits in the circuit's output data. Chapter VI also demonstrates that RPR requires much less circuit area and power than TMR designs with identical latency and throughput performance. This data can be combined with the parameter scaling factors *K* to determine optimal design solutions to real-world problems, as described in the satellite image processing example earlier in this chapter.

THIS PAGE INTENTIONALLY LEFT BLANK

V. CORDIC ALGORITHM

A. OVERVIEW

The CORDIC algorithm was chosen as a test case for exploring the effectiveness of the fault-tolerant and power-saving concepts from Chapters II and III. In order to investigate these issues in a realistic scenario, a CORDIC-based sine/cosine calculator was constructed for Xilinx Virtex FPGAs. Several designs were created that implemented both iterative and pipelined versions with varying precision. The details of these implementations are provided in Appendix A. These designs incorporated fault-tolerant features to make them reliable and suitable for spacecraft computer systems. As discussed in Chapter II, the most common fault-tolerant methods for FPGAs employ spatial redundancy. TMR and RPR designs using the basic CORDIC processor were built for the CFTP system to be launched into low-earth orbit in late 2006. The designs were then tested in the lab for SEU fault tolerance and power consumption. They were also made flexible to permit exploration of the trade space between fault tolerance, power usage, and other performance parameters described in Chapter IV. For example, the precision of the upper/lower bound calculations in RPR designs can be varied easily. Changing the precision of these error bounds affects area, power, precision and reliability. Therefore, many design options can be examined using the same basic algorithm and architecture. This chapter describes the CORDIC algorithm and why it is useful for investigating trade-offs between TMR and RPR fault-tolerant approaches.

1. Rationale for Choosing CORDIC

CORDIC is a good test case for numerous reasons. It is a well-known algorithm founded on basic mathematical principles and widely used in a variety of applications. A self-contained CORDIC module can be used in a stand-alone configuration or as a coprocessor for larger systems such as satellite control computers. The algorithm can be readily implemented on FPGAs [25], [34], [90]. Since most implementations include a mixture of combinational and sequential logic, it is representative of general complex circuits. Thus, faults in a CORDIC design can be manifested in more complicated ways than faults in a trivial circuit. In addition, CORDIC circuits can be built as either iterative

or pipelined designs of varying wordlength in order to balance criteria such as throughput, size, and accuracy. Such designs can be easily scaled to occupy the desired fraction of an FPGA to effectively investigate power usage and fault tolerance.

A CORDIC processor better reflects the kinds of circuits used in real-world systems because it has more complicated input/output relationships than simpler circuits, such as the small binary counters used in [29]. The feedback inherent in the iterative CORDIC method presents interesting error propagation behavior not observed in simple circuits. The iterative design can be viewed as a fairly complex finite state machine, raising questions about when and where results should be checked for correctness (e.g., at every iteration or only at the final result?). Though relatively complex, a CORDIC design is more manageable in terms of size and complexity than extremely complicated circuits such microprocessors, and can be easily scaled to meet the requirements of a wide range of applications.

Another important characteristic of the CORDIC algorithm is that it fits the criteria of Class A given in Chapter II. Step 2 in the flowchart from Figure 2.12 asks whether the system can occasionally accept lower precision results. The answer is affirmative for many CORDIC applications that can accept "flexible precision," such as signal processing and control systems. CORDIC also meets the criteria of steps 3 and 4 in the flowchart, which ask whether approximate solutions can be formulated and are more efficient. By changing the wordlength and/or number of iterations in a CORDIC processor, the precision of the solutions can be adjusted. A less precise processor will necessarily be smaller and therefore more efficient in terms of area, time and power. Thus, CORDIC is a good candidate for the RPR fault-tolerant technique.

2. CORDIC Applications

Although the original use of CORDIC was in airborne navigation [91], the CORDIC technique can calculate numerous functions for a broad range of applications. It can be used to calculate transcendental functions (sine, cosine, arctangent, exponential, etc.), coordinate transformations (Cartesian-to-polar, etc.), and other common arithmetic

functions (multiplication, division, square root, etc.) [30]. CORDIC processors have been used for applications as varied as handheld calculators, radar signal processing, and image processing [90].

CORDIC has been used in many DSP applications, such as discrete Fourier transforms and matrix solvers [25]. It has also been used to calculate the Discrete Cosine Transform (DCT), which is widely used for image compression and other signal processing tasks. Numerous papers describe efficient CORDIC-based DCT techniques [92], [93].

One particularly intriguing paper investigates low-power image processing using a CORDIC vector interpolator [65]. The authors achieve over 70% reduction in power consumption by adjusting the precision of the calculations to match the dynamically changing scene. This variable precision calculation is enabled by controlling the number of CORDIC iterations performed – higher precision uses more iterations and more power, while lower precision requires less power.

CORDIC has proven to be an efficient and powerful computing technique for VLSI designs and, more recently, FPGA implementations [25], [94]. In fact, CORDIC is so widely used that it is now available as intellectual property (IP) "cores" from Xilinx [95] and other vendors. Thus, studying CORDIC designs has considerable practical utility.

B. MATHEMATICAL FOUNDATION

The CORDIC computational technique is an elegant method for calculating a surprising number of functions using simple circuit elements. Volder developed this method to calculate navigation equations in a real-time aircraft computer system [91]. He named his system the COordinate Rotation DIgital Computer (CORDIC) and first published the technique in 1959. Its basic formulation is an iterative algorithm based on simple trigonometric relationships that describe vector rotation in 2-dimensions. It was later generalized to include 6 possible modes of operation {circular rotation/vectoring, linear rotation/vectoring, hyperbolic rotation/vectoring} that enable the calculation of numerous standard and transcendental functions [90]. One of the more commonly used modes is the circular rotation mode, which is derived in the next section.

1. Derivation of Circular Rotation Mode

The basic concept for the circular rotation mode can best be described using Figure 5.1. A unit length vector originally aligned along the x-axis is rotated by an angle θ. Since the vector length remains equal to 1 through this rotation, the final *x* and *y* values give the cosine and sine values of the angle θ.

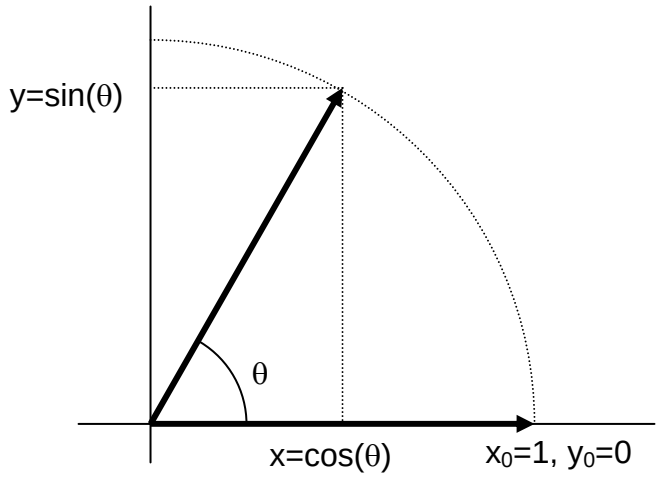


Figure 5.1 Vector Rotation in 2 Dimensions

In the CORDIC algorithm, this rotation process is performed as a series of pre-defined rotation steps of decreasing magnitude that add up to the total desired rotation angle. These rotation steps are chosen to be $\alpha_i = \pm\{45°, 26.6°, 14.0°, 7.1°, \ldots\}$ according to the formula $\alpha_i = \pm\text{atan}(2^{-i})$, as discussed in the next section. For example, 10° of rotation can be approximated in four steps as θ = 45 – 26.6 – 14 + 7.1 = 11.5°. Any angle can be approximated to the desired precision using the appropriate number of steps.

However, if this "true" rotation is followed, computing the x and y values at each step requires sine and cosine calculations. Therefore Volder devised the concept of a pseudorotation, in which the rotated vector is extended to meet a line perpendicular to the original vector. A pseudorotation spans the same angle as a true rotation, but causes the vector length to increase. Figure 5.2 shows the geometry of these two methods for rotating the vector (x_1,y_1) by α.

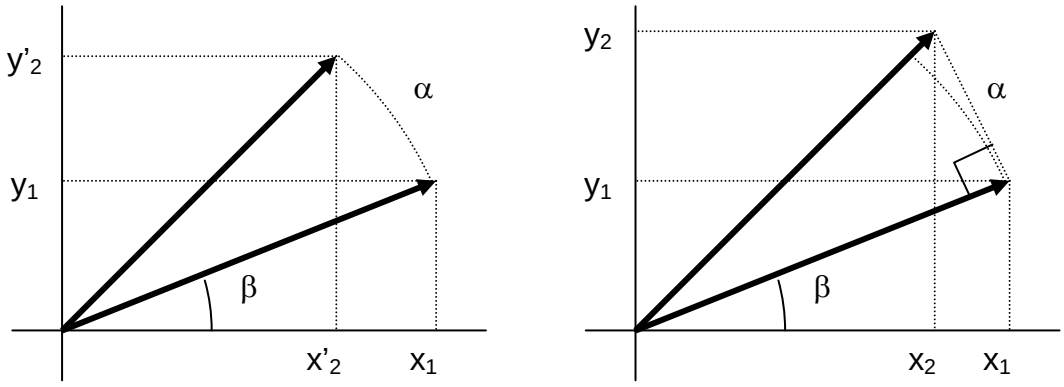


Figure 5.2 True Rotation (left) versus Pseudorotation (right)

For a vector of length R, the original vector components are given by:

$$\begin{aligned} x_1 &= R\cos\beta \\ y_1 &= R\sin\beta \end{aligned} \tag{5.1}$$

while the rotated vector components in true rotation are given by:

$$\begin{aligned} x'_2 &= R\cos(\alpha+\beta) = x_1\cos\alpha - y_1\sin\alpha \\ y'_2 &= R\sin(\alpha+\beta) = x_1\sin\alpha + y_1\cos\alpha \end{aligned} \quad \text{[True]} \tag{5.2}$$

and the vector in pseudorotation is given by:

$$\begin{aligned} R_2 &= R/\cos(\alpha) = R\sqrt{1+\tan^2\alpha} \\ x_2 &= R_2\cos(\alpha+\beta) = x'_2\sqrt{1+\tan^2\alpha} \\ &= x_1 - y_1\tan\alpha \\ y_2 &= R_2\sin(\alpha+\beta) = y'_2\sqrt{1+\tan^2\alpha} \\ &= y_1 + x_1\tan\alpha \end{aligned} \quad \text{[Pseudo]} \tag{5.3}$$

These equations form the basis for the CORDIC algorithm. At each step, *i*, the vector is rotated by a pseudorotation angle α_i, as shown in the following equations. The variable λ tracks the remaining amount of angular rotation.

$$\begin{aligned} X_{i+1} &= X_i - Y_i\tan\alpha_i \\ Y_{i+1} &= Y_i + X_i\tan\alpha_i \\ \lambda_{i+1} &= \lambda_i - \alpha_i \end{aligned} \tag{5.4}$$

Unlike true vector rotations, the CORDIC process does not preserve the length of the original vector. Each pseudorotation lengthens the vector by the factor $\sqrt{1+\tan^2\alpha}$ in Equation 5.3 above. This lengthening is independent of whether the rotation α is positive or negative. The total vector lengthening is a constant, K, that depends only on the number of iterations performed. With the standard angular steps of $\pm\text{atan}(2^{-i})$ described in the next section, for 10 or more iterations *K* is roughly 1.646760. Since the vector expansion factor can be determined *a priori*, one can compensate by appropriately scaling the original vector (X_0, Y_0).

2. Selection of Pseudorotation Angles

For ease of implementation in a digital system, the values of $\tan(\alpha)$ are restricted to powers of two (2^0, 2^{-1}, 2^{-2}, …). This simplification allows the multiplications that appear in Equation 5.4 above to be treated as simple right shifts of the binary numbers. Therefore, the increments by which λ can change are restricted to the values $\pm\text{atan}(2^{-i})$.

i =	0	1	2	3	4	5	6	…
α_i (deg) =	45	26.6	14.0	7.1	3.6	1.8	0.9	⋯
α_i (rad) =	0.785	0.464	0.245	0.124	0.062	0.031	0.016	⋯

Table 5.1 Pseudorotation Angles for Circular CORDIC Modes

Finally, an additional term is needed to determine whether the rotation increment at each step should be positive (counter-clockwise) or negative (clockwise). The variable $\xi_i \in \{1, -1\}$ tracks these rotation directions to ensure the vector approaches the desired final angle. The fundamental CORDIC equations are therefore:

$$\begin{aligned} X_{i+1} &= X_i - \xi_i Y_i 2^{-i} \\ Y_{i+1} &= Y_i + \xi_i X_i 2^{-i} \\ \lambda_{i+1} &= \lambda_i - \xi_i \tan^{-1} 2^{-i} \\ \xi_i &= \text{sign}(\lambda_i) \quad \text{or} \quad = -\text{sign}(X_i Y_i) \end{aligned} \tag{5.5}$$

These are the forms of the equations needed for implementation in a digital system. The essential hardware elements are 3 registers, 3 add/subtract units, 2 shifters, a sign detector, and a lookup table. Figure 5.3 shows how these components are arranged in an iterative implementation. Details such as register reset signals, LUT address lines, output rounding and other input/output processing are not shown.

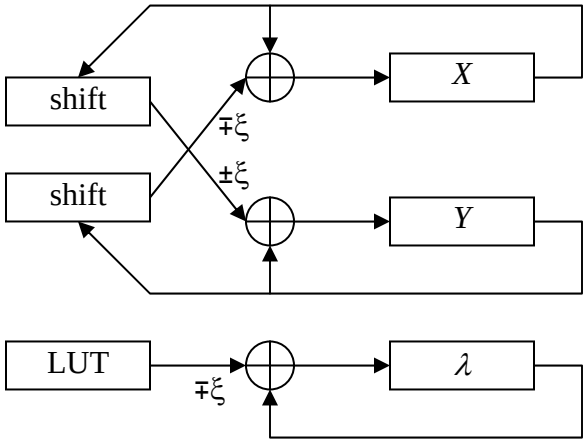


Figure 5.3 Iterative CORDIC Hardware Configuration

Depending on which sign formula is used for ξ, different functions can be calculated. Choosing the first formula (rotation mode) and setting (X_0=1/K, Y_0=0) produces cosine and sine functions. After m iterations, the values X_m and Y_m represent approximations of the cosine and sine of the starting angle λ_0. With the second formula (vectoring mode) and λ_0=0 the function atan(Y_0/X_0) is produced.

C. ERROR PROPAGATION IN ITERATIVE AND PIPELINED DESIGNS

Although a CORDIC processor is comprised of simple circuit elements, the iterative nature of its calculations causes errors to propagate in complicated ways. This complex behavior can cause single bit errors in the input or internal circuit to cascade into multiple data bit errors at the output. This cascade of errors can happen in both iterative and pipelined implementations. Thus, standard error detection and correction schemes are insufficient for coping with faults in CORDIC designs and other circuits with similar iterative structures.

1. Feedback and Error Propagation

An iterative CORDIC circuit has quite obvious feedback paths, since the (X, Y, λ) register values on one clock cycle are used to calculate the same register values on the next clock cycle. Depending on when and where a fault occurs, it can corrupt a large fraction of the output data bits. In addition, the control function is a finite state machine that makes decisions about when to start/stop the iterations and keeps track of the iteration number. While some failures in this controller have mild consequences, other failures can be catastrophic.

By contrast, a pipelined CORDIC design does not contain such feedback elements. The pipeline is entirely made up of feed-through logic. There is no need to keep track of iteration counts since the lookup table values can be distributed along the pipeline and the shifting function can be hardwired. Nonetheless, some faults can still lead to multiple bit errors at the output. For example, SEU-induced configuration faults in the early pipeline stages have a greater numerical impact on the result (affecting bits closer to the MSB) than faults in later pipeline stages (affecting bits closer to the LSB).

Two brief examples demonstrate typical error propagation behavior. Both cases produce outputs with 8-bit resolution using the two's complement fixed-point number scheme from Appendix A (Appendix B contains the MATLAB code that was used for generating these examples). Eight iterations and an 11-bit internal datapath are needed to provide accurate 8-bit output values. Appropriate rounding, not shown in this section, must be applied to the final X and Y values between the internal registers and the output lines. These examples use the circular rotation mode to produce sine and cosine values of the input angle of 30° (0.5236 radians). Thus the correct answers should be $X = \cos(30°) = \sqrt{3}/2 = 0.866$ and $Y = \sin(30°) = 1/2 = 0.500$.

Table 5.2 shows what happens when an SEU upsets bit #4 of the Y register during iteration #6 in an iterative design. As expected, the final Y value is significantly corrupted, whereas the final X value is changed only slightly. However both answers have 2 bits in error. Note that the angle values are unaffected, as the λ datapath has no dependence on the X and Y values for this CORDIC mode.

	i	Fault-free		Faulty	
	0	00100110111	= 0.607	00100110111	= 0.607
	1	00100110111		00100110111	
	2	00111010010		00111010010	
	3	00110101011		00110101011	
X	4	00111001101		00111001101	
	5	00111000000		00111000000	
	6	0011011***0***01		0011011***1***01	
	7	0011011***0***1		0011011***1***1	
	8	0011011***00***	= 0.867	0011011***11***	= 0.873
	0	00000000000	= 0.000	00000000000	= 0.000
	1	00100110111		00100110111	
	2	00010011100		00010011100	
	3	00100010000	Fault occurs here	00100010000	
Y	4	00011011011		00011011011	
	5	000***1***1110111		000***0***1110111	
	6	00***10***0000101		00***01***0000101	
	7	000***1***1111111		000***0***1111111	
	8	00***10***0000010	= 0.504	00***01***0000010	= 0.254
	0	00100001100	= 0.527	00100001100	= 0.527
	1	11101111010		11101111010	
	2	00001100111		00001100111	
	3	11111101010		11111101010	
λ	4	00000101010		00000101010	
	5	00000001010		00000001010	
	6	11111111010		11111111010	
	7	00000000010		00000000010	
	8	11111111110	= -0.004	11111111110	= -0.004

Table 5.2 Example Error Propagation in Iterative CORDIC

The second scenario involves an SEU causing a configuration fault in the carry logic for the third stage adder along the λ datapath of a pipelined design. This hypothetical fault causes an inversion of the carry signal from bit position 6. As seen in the fourth λ row of Table 5.3, the carry propagation error has a ripple effect on the 5 leftmost bits. Because the X and Y calculations depend on the sign of the angle value, this fault causes significant data corruption in both the sine and cosine results. The final *X* and *Y* values each contain 3 bit errors. Interestingly, the angle value error is corrected in subsequent stages and by the fifth stage recovers completely to the fault-free value. However, this correction of the angle value only occurs in the rotation modes. In the vectoring modes, an error in the λ register would persist since the iterations are designed to drive the *Y* value to zero.

	Stage	Fault-free		Faulty	
	0	00100110111	= 0.607	00100110111	= 0.607
	1	00100110111		00100110111	
	2	00111010010		00111010010	
	3	00110101011		00110101011	
X	4	00111001101		00110001001	
	5	00111000000		00110011101	
	6	00110111001		00110010100	
	7	00110111101		00110011000	
	8	00110111100	= 0.867	00110010110	= 0.793
	0	00000000000	= 0.000	00000000000	= 0.000
	1	00100110111		00100110111	
	2	00010011100		00010011100	
	3	00100010000		00100010000	
Y	4	00011011011		00101000101	
	5	00011110111		00100101101	
	6	00100001001		00100111001	
	7	00011111111		00100110011	
	8	00100000010	= 0.504	00100110110	= 0.605
	0	00100001100	= 0.527	00100001100	= 0.527
	1	11101111010		11101111010	
	2	00001100111		00001100111	
	3	11111101010		00000101010	
λ	4	00000101010	Fault occurs here	11111101010	
	5	00000001010		00000001010	
	6	11111111010		11111111010	
	7	00000000010		00000000010	
	8	11111111110	= -0.004	11111111110	= -0.004

Table 5.3 Example Error Propagation in Pipelined CORDIC

The significance of these observations is that modular redundancy techniques such as TMR and RPR are essential for protecting against faults in FPGA implementations of CORDIC. Error detection and correction (EDAC) codes, such as Hamming and Reed-Solomon codes, are effective for fixing a small percentage of bit flips during data transmission or storage, but are incapable of correcting large numbers of bit errors that can occur due to functional faults [30]. EDAC methods generally operate by checking input/output data against a "dictionary" of valid codewords, focusing on faults that affect the data bits themselves instead of faults in the underlying operation of the circuit. Some coding techniques, such as residue coding, provide tolerance against functional faults within circuits such as simple adders [96]. However, the properties of residue codes are only preserved in addition, subtraction and multiplication [97] so residue coding is not suitable for the division operations inherent in the CORDIC

iterations. Techniques such as TMR and RPR overcome these limitations and are more effective for protecting complex circuits such as CORDIC.

2. Using TMR and RPR to Correct CORDIC Faults

The effectiveness of TMR and RPR can be demonstrated using the preceding examples from Table 5.2 and Table 5.3. The TMR approach uses three identical copies of the processing module, whereas RPR uses one copy of the processing module and two smaller modules for calculating the upper and lower bounds. Figure 5.4 shows a block diagram of how RPR would be implemented. In these examples, the input and output datawidth for the exact module is 8-bits (n=8, h=8), though the internal datapath is 11-bits wide. For these examples, assume that 5-bit calculations of upper and lower bounds are sufficient (m=5, k=5).

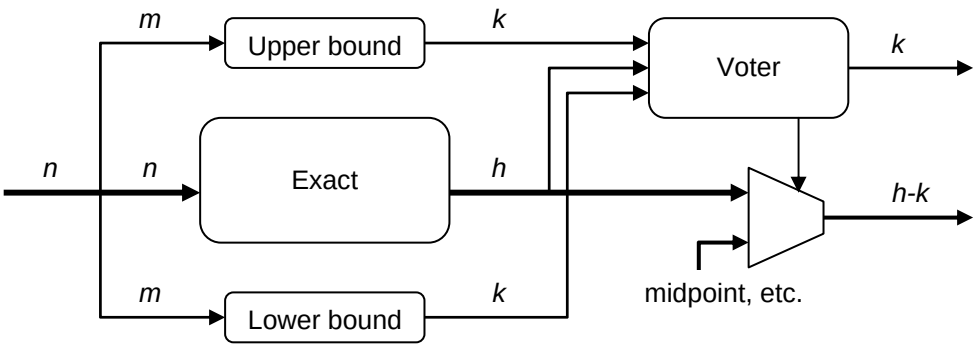


Figure 5.4 General RPR Configuration

With these assumptions, the behavior of both TMR and RPR designs can be predicted. Considering only a single fault that affects one of the identical TMR modules or the exact module in RPR, the voters in both designs are presented with the choices given in Table 5.4 and Table 5.5 for the two fault scenarios described in the previous section. Note that the full-precision values from Table 5.2 and Table 5.3 have been rounded to 8 bits , as mentioned in Section 1.

TMR			RPR		
Module A (fault-free)	*X*	00111000	Module A "upper" (fault-free)	*X*	00111
	Y	00100000		*Y*	00101
	λ	00000000		λ	00000
Module B (faulty)	*X*	00111000	Module B "exact" (faulty)	*X*	00111000
	Y	00***01***0000		*Y*	00***01***0000
	λ	00000000		λ	00000000
Module C (fault-free)	*X*	00111000	Module C "lower" (fault-free)	*X*	00110
	Y	00100000		*Y*	00100
	λ	00000000		λ	11111
Voter	*X*	00111000	Voter	*X*	00111000
	Y	00100000		*Y*	00100---
	λ	00000000		λ	00000000

Table 5.4 Example Response of Iterative TMR and RPR Designs

As expected, in both scenarios the TMR version produces the correct answer with full precision since it has access to two fault-free answers. RPR is more complicated because its fault response depends upon how far the exact solution deviates from the correct value. In the iterative case above, the rounded value of the *X* result is the same for the faulty and fault-free modules, thus neither voter sees any conflict. However, the *Y* result error is detected by the voters. The RPR voter recognizes that the faulty *Y* value is outside the range of the upper/lower bounds and must make a decision about what value to report. As shown in the table, the voter reports 00100---, with "-" indicating that these digits may differ depending on the particular implementation. For example, a logical choice might be to provide a "midpoint" value like that suggested in Figure 5.4. Thus 00100--- would become 00100100.

In the pipelined case below, the data errors exist in the least significant bits of the *X* and *Y* values. Because this particular circuit fault yields only small numerical errors, the *X*, *Y*, and λ values from the exact module all fall within the bounds of RPR modules A and C. Therefore the voter doesn't detect any problem and passes along the results from module B. As indicated in the table, this permits three erroneous bits to propagate. However, all of the results are within the tolerance range of a 5-bit RPR design ($\pm 2^{-(5-2)} = \pm 0.125$). This possibility of imprecise results is the trade-off that permits savings in area and power when implementing RPR fault tolerance.

TMR			RPR		
Module A (fault-free)	*X*	00111000	Module A "upper" (fault-free)	*X*	00111
	Y	00100000		*Y*	00101
	λ	00000000		λ	00000
Module B (faulty)	*X*	0011***0***0***11***	Module B "exact" (faulty)	*X*	0011***0***0***11***
	Y	00100***111***		*Y*	00100***111***
	λ	00000000		λ	00000000
Module C (fault-free)	*X*	00111000	Module C "lower" (fault-free)	*X*	00110
	Y	00100000		*Y*	00100
	λ	00000000		λ	11111
Voter	*X*	00111000	Voter	*X*	0011***0***0***11***
	Y	00100000		*Y*	00100***111***
	λ	00000000		λ	00000000

Table 5.5 Example Response of Pipelined TMR and RPR Designs

It is important to point out that neither redundancy technique is infallible. TMR and RPR are vulnerable to faults that affect any single point of failure, such as non-replicated voter components and SEFI-type vulnerabilities unique to FPGAs. In addition, they are both vulnerable to multiple faults that simultaneously affect more than one module. However, when using FPGA configuration scrubbing, the likelihood of experiencing SEU-induced faults in multiple modules at the same time is extremely rare.

Chapters VI and VII demonstrate the effectiveness of these two redundancy approaches for various CORDIC implementations. Of particular interest are the relative effectiveness and efficiency of these two techniques. By quantifying the SEU susceptibility and area/power usage of RPR and TMR, one can make informed decisions about which method is most appropriate for a particular real-world problem.

THIS PAGE INTENTIONALLY LEFT BLANK

VI. SIMULATION ENVIRONMENT AND RESULTS

A. OVERVIEW

In order to demonstrate the effectiveness and efficiency of the RPR fault tolerance approach, simulations were conducted to determine the SEU sensitivity and power consumption of several CORDIC designs implemented on FPGAs. SEU simulations were performed using hardware and software tools developed at NPS for the CFTP program, and enhanced to support this research. The completion of this SEU simulation system is a major contribution of this work, as it allows comprehensive ground-based testing for emulating and predicting performance of the CFTP experiments in their operational space environment. The SEU simulator was validated using radiation test data from experiments conducted at UC Davis' Crocker Nuclear Lab (see Chapter VII). Power estimates for the CORDIC designs were made using commercial software tools, following the methodology used at BYU [9]. Predictions of power consumption were made by integrating high-fidelity timing simulations of the circuits with manufacturer data of FPGA circuit parameters.

The data presented in this chapter supports the assumptions made in previous chapters regarding the benefits of an RPR fault-tolerant design. Compared to the TMR designs tested in this research, the RPR designs show comparable SEU tolerance but require significantly less power. Thus for many applications, especially those in which power is a significant constraint, the RPR architecture is superior to the typical TMR approach.

B. SEU SIMULATIONS

1. SEU Simulation Environment

As part of the CFTP research program, a fault injection system was built at NPS to simulate the effects of SEUs on FPGA circuits [98]. The initial fault injection system consisted of manual command-line and graphical user interface methods of forcing single-bit upsets into an operational FPGA configuration bitstream. The command-line method was used to simulate upsets to specific bits and/or regions on the device. The

graphical interface injected faults in a more random manner, but provided visual confirmation of whether or not those faults appeared in regions of the FPGA occupied by the functional circuit. The effects of the simulated SEUs were determined using automatic error detection circuits programmed on the FPGAs, by visual monitoring of the output data stream, and through post-processing of the data. These manual techniques were used in preparation for radiation testing and in analysis of the radiation test results.

As part of this research, the manual methods were later augmented with an automatic fault injection system that dramatically improved the speed and capability of the SEU simulator. The CFTP hardware configuration is well-suited for this task because it includes two Xilinx Virtex FPGAs and an on-board Flash memory device for holding configuration bitstream data. Figure 6.1 shows the major components in the SEU simulator configuration. The first FPGA serves as the controlling device and the second FPGA is the experiment device.

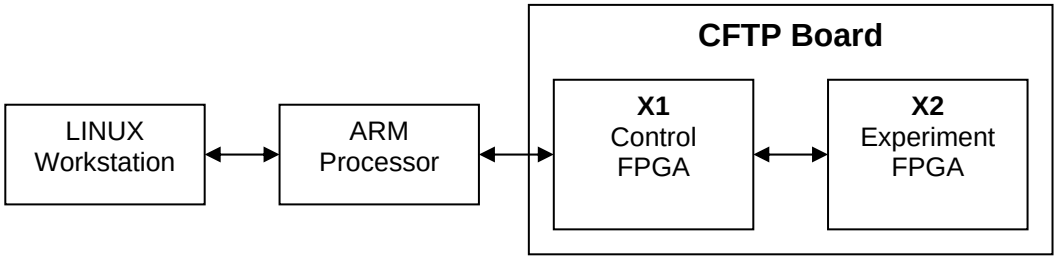


Figure 6.1 SEU Simulator Configuration with CFTP Hardware

The automated SEU simulator tests the SEU sensitivity of the circuit design loaded onto the experiment FPGA. The control FPGA reads fault-free configuration data from the Flash memory, programs the experiment FPGA with artificially corrupted bitstream data, and reports the effect of each simulated SEU on the experiment FPGA output data. Since some configuration faults only manifest themselves as data errors for certain input values, it is important for the X2 circuit to process input vectors that exercise as many circuit paths as possible. The normal simulator settings allow the X2 circuit to run for approximately 1 msec after each bit is toggled to capture these data-dependent sensitivities. The CFTP board has a 50 MHz clock, but throughput is lower. For example, the 32-bit iterative CORDIC circuits described below divide the clock down to 6.25 MHz and require 37 cycles to produce each result. Thus, these circuits

process 168 unique inputs in a 1 msec period. In addition, pseudo-random number generators are used to produce input vectors for the X2 circuit to maximize the chance of discovering error-producing configuration faults. Data is collected by the ARM processor (also part of the CFTP flight hardware) and sent to the LINUX workstation for storage and analysis.

Automation is essential for enabling the comprehensive measurement of SEU sensitivity for complex FPGA circuits in which millions of configuration bits must be individually tested. Even with the automation of the fault injection process, comprehensive SEU simulations are a time-consuming endeavor. In normal operation, the SEU simulator takes over one hour to exhaustively test all 3.3 million configuration bits on the Virtex XQVR600 device. Designs with very small sensitivities have few errors to report and the total process finishes in slightly over 1 hour. Designs with greater SEU sensitivity take longer to run because the error reports slow the process. For example, a design with about 40,000 sensitive bits requires nearly 1.75 hours to complete.

Although the manual fault injection methods are not practical for comprehensive SEU sensitivity studies, they were invaluable in developing and verifying the automated fault injection process. The initial fault injection methods were used for rapid confirmation of radiation results. Data from these manual simulations and radiation tests were compared with results from the automated simulator, thus demonstrating the validity of the automatic method of simulating radiation-induced SEUs. During radiation testing the proton flux was controlled to ensure that the exact bits causing data errors could be easily isolated. These configuration bits were then manipulated in the simulator to verify that they were responsible for the data errors observed in the radiation environment. Chapter VII discusses the details of the radiation testing and presents results from the verification effort. Once validated, the SEU simulator can be used confidently to examine the sensitivity of FPGA circuits without expensive and time-consuming testing in a radiation facility. The following sections demonstrate the effectiveness of the RPR method through simulations of various CORDIC test circuits.

2. Test Circuits

Several CORDIC circuits were tested to draw fair and accurate conclusions about the comparative reliability of TMR and RPR approaches. The basic CORDIC design is detailed in Appendix A. However, some important details differ between the various test designs shown below. The first two test circuits, labeled "Davis TMR" and "Davis RPR," correspond to the exact circuits used during radiation testing. Several improvements implemented after the UC Davis test runs in Nov 2005 included: 1) replicating the input vector counters and assigning each TMR or RPR module its own counter, 2) using LFSRs for input number generators, and 3) simplifying the TMR voter logic to operate as a bit-wise voter rather than vector-wise. Changing the input vector counters from simple binary counters to LFSR pseudo-random number generators provided better coverage of the input vector space. This is because practical issues (in particular, the time required to test a circuit) limit the number of inputs that can run through the circuit for each simulated SEU. The LFSR approach offers a better sampling of the range of possible input values.

Finally, it is important to note that the TMR designs built for these tests differ somewhat from the TMR style suggested by Xilinx [68]. Instead they follow more closely with the TMR designs tested in [85]. Whereas Xilinx's approach involves triplication of logic functions as well as all clock signals and input/output pins, the TMR circuits tested here do not replicate the clock and I/O signals. Rather, a single set of inputs are shared among all modules and all data results are voted within the FPGA before being output. The following sections describe the unique features of each test circuit. Different and more extensive TMR, RPR and other techniques can easily be inserted in the simulation testbed developed here to support future research.

a. "Davis TMR" Iterative CORDIC

Figure 6.2 shows the layout for the TMR circuit used during radiation testing at UC Davis (note that the cloud shapes reflect the fact that the component placement was not specified as a design constraint and instead the Xilinx tools were allowed to automatically place and route the circuit). It consists of three copies of an iterative CORDIC processor computing the sine and cosine functions, as detailed in

Appendix A. Inputs and outputs are 32 bits wide, while internal computations are performed with 39-bit precision. A single binary counter is included to generate 32-bit input angle values for all three modules. Although three different results are computed in each module (sine, cosine, and residual angle), due to I/O pin limitations only the sine value is compared and output from the FPGA. The only input to the FPGA is a reset signal that is used to synchronize the X1 and X2 devices by initializing all registers, including the angle input counter. Outputs from the device consist of the voted 32-bit sine value and an error flag indicating whether the TMR voter detected any mismatch among the three modules.

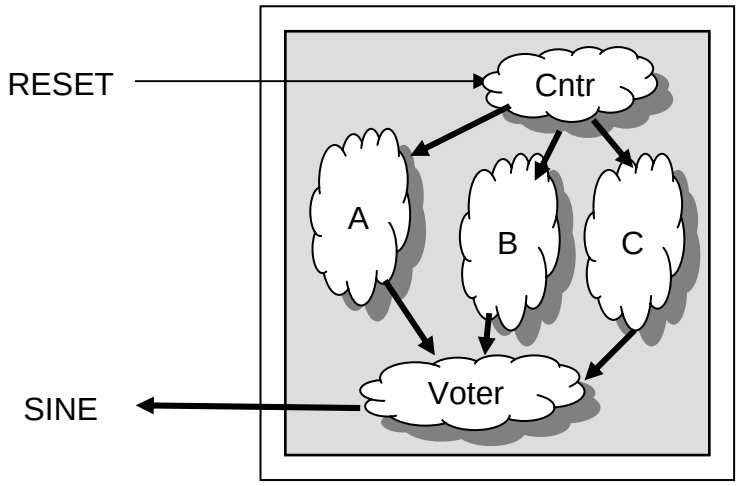


Figure 6.2 Layout for "Davis TMR" Circuit

b. "Davis RPR" Iterative CORDIC

The RPR circuit used during radiation testing is sketched in Figure 6.3. It consists of a single copy of the 32-bit iterative CORDIC processor used in the "Davis TMR" design, as well as 8-bit upper and lower bounds calculations that are implemented as simple look-up tables. These upper/lower bounds provide protection against faults that cause the full-precision module to produce grossly inaccurate results, as is the intent with the RPR approach. A single binary counter feeds input vectors to all three modules and a reset signal synchronizes the FPGAs. Like the TMR circuit, only the sine results are compared and output. Also, an error flag tells when the voter detects a result from the precise module that falls outside the range of the upper/lower bounds.

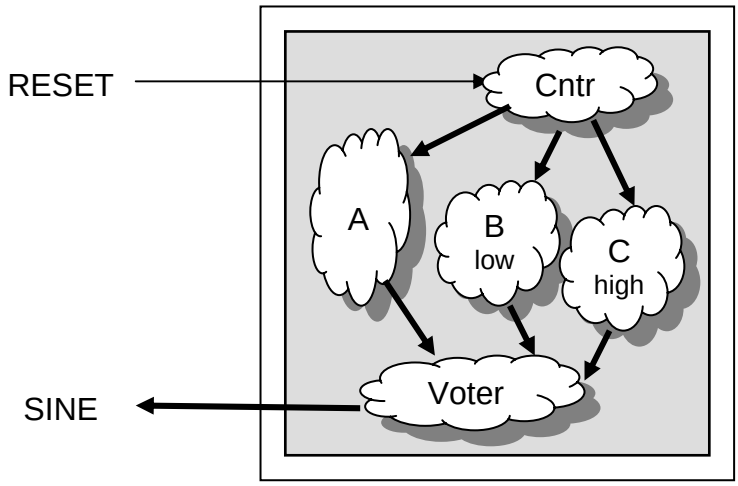


Figure 6.3 Layout for "Davis RPR" Circuit

c. "Unprotected" Iterative CORDIC

In addition to the TMR and RPR designs, circuits without any fault tolerant features were tested. The basic configuration for these circuits is shown in Figure 6.4. The first of these circuits was a 32-bit iterative CORDIC processor based on the VHDL design given in Appendix A. The input and output behavior of this circuit is identical to the Davis circuits in a fault-free situation. An additional unprotected circuit with only 16-bits of computational precision was tested to understand the correlation between circuit size and SEU sensitivity. These circuits have no way of mitigating SEUs that corrupt their configuration bits. Testing an unprotected circuit is useful for determining a baseline failure rate, which can then be compared with fault tolerant design options. The benefit of the fault tolerant approaches can be assessed relative to the reliability of the unprotected design.

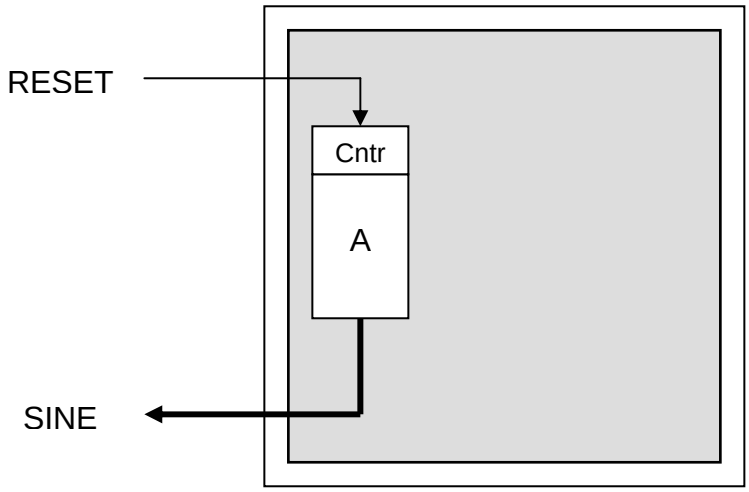


Figure 6.4 Layout for "Unprotected" Circuits

d. "Improved TMR" Iterative CORDIC

Following the radiation testing at UC Davis, some deficiencies were identified in the overall structure of the TMR and RPR designs. Therefore several improvements were made to the original designs. As explained in Appendix A, while the original "Davis" circuits were generated mostly with schematic design tools, subsequent circuits were created using VHDL. This permitted more rapid adjustment of numerical precision, voter design, and other parameters. TMR versions of the iterative CORDIC circuits were tested with various degrees of precision. As shown in Figure 6.5, the improved TMR circuits include separate input number generators for each CORDIC module in order to eliminate the input counter as a single point of failure. In addition, LFSRs are used instead of simple binary counters to improve the randomness of the circuit input values. Finally, the TMR voter was changed so that it operates on each output bit individually instead of the entire output data word. Voting each bit separately is more common in TMR designs. Also note that the rectangular shapes reflect the explicit circuit placement constraints that were used to ensure that faults in a certain region of the device did not affect more than one module.

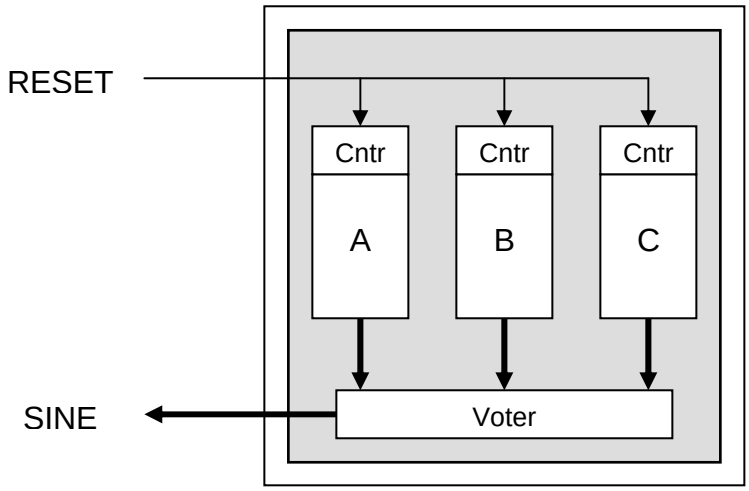


Figure 6.5 Layout for "Improved TMR" Circuits

e. "Improved RPR" Iterative CORDIC

Following the same rationale described for the "improved TMR" designs, some improvements were made to the original RPR designs tested at UC Davis. The "improved RPR" designs were created in VHDL, include replicated LFSR input number generators, and have enhanced voter layout. These circuits maintain the basic RPR architecture of computing a full precision solution and lower precision upper and lower bounds, as shown in Figure 6.6. One key enhancement to the voter is a checker that detects errors in the numerical difference between the lower and upper bounds. This checker ensures that the upper bound is not less than or equal to the lower bound. In addition, the checker confirms that the difference between the upper and lower bounds is less than or equal to the maximum allowed. Based on the rounding methods used here, this difference should be no more than one digit in the least significant bit in the bounds calculations.

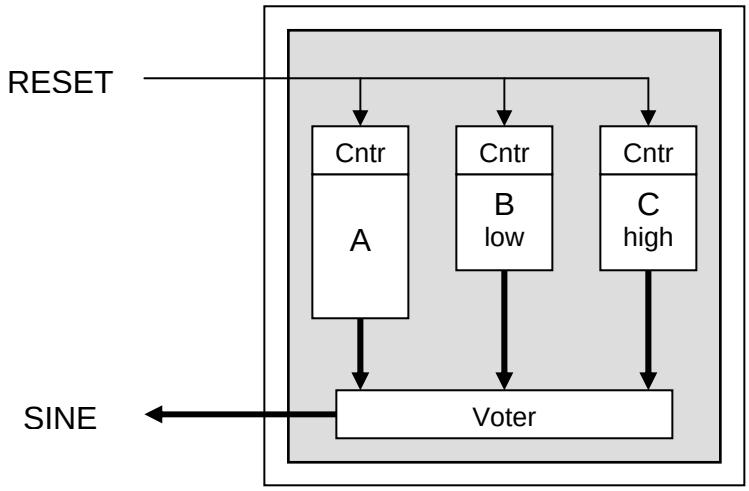


Figure 6.6 Layout for "Improved RPR" Circuits

3. Results

a. Data

The fundamental product of the SEU simulator is the number of configuration bits in a circuit design that cause errors in output data. These bits are referred to as "sensitive" [42]. A circuit's overall sensitivity is expressed either as a total count of sensitive bits or as a fraction of sensitive configuration bits. Such data is directly related to the circuit's dynamic SEU cross section, a term used elsewhere in the literature [85] (the word "dynamic" indicates that a functional circuit must be actively operating to determine this sensitivity). Multiplying an FPGA's static cross section by a circuit's sensitivity fraction yields the dynamic cross section of a circuit. This value can then be multiplied by the predicted radiation flux in space to arrive at an expected failure rate in the operational environment.

Table 6.1 summarizes the results from SEU simulations on the CORDIC circuits described in Section 2 above. As explained in Section 1, the experiment and control FPGAs were synchronized so that the data from the X2 FPGA could be verified against the fault-free computation being performed simultaneously on X1. The data in Table 6.1 gives the number and fraction of configuration-bit SEUs that produce at least one erroneous data value during the simulator's 1 msec error detection interval. These are data errors that could not be corrected by mitigation methods on X2. Thus low counts are desirable.

CORDIC Design Name (Virtex XQVR600cb228-4) (3,378,600 config bits)	Area (Slices)	# Sensitive Bits	Fraction Sensitive Bits
Davis TMR Iterative	1326	30,604	0.91%
Davis RPR Iterative	540	31,837 (350)*	0.94% (0.01%)*
Unprotected 32-bit Iterative	843	66,190	1.96%
Unprotected 16-bit Iterative	238	21,065	0.62%
Improved 32-bit TMR Iterative	2541	30,003	0.89%
Improved 16-bit TMR Iterative	667	11,616	0.34%
Improved 32-bit RPR Iterative	933	65,699 (22,691)*	1.94% (0.67%)*
Improved 16-bit RPR Iterative	309	22,922 (15,291)*	0.68% (0.45%)*

Table 6.1 SEU Simulator Results for Uncorrected Errors
()* Errors in 8 Upper Bits Only

Although not shown in Table 6.1, the data collected during SEU simulations also includes the total number of erroneous outputs accumulated during the 1 msec interval that each SEU is allowed to persist. Some bit upsets lead to data errors on every calculation, while other upsets only corrupt the output data for certain inputs. During the 1 msec error detection interval, the 32-bit CORDIC circuits process 168 unique input values. Thus upsets to the most critical bits cause a maximum error count of 168, whereas less troublesome bit upsets cause a smaller number of errors. These error counts can be interpreted as the probability of circuit failure when an SEU affects a particular bit, as described in [99]. The histogram in Figure 6.7 shows an example of the distribution of sensitive configuration bits in terms of this failure probability, using data from the "Davis TMR" circuit. Simulation results for the various test circuits show that the vast majority of the error-producing SEUs affect the output for most or all of the input values. Similar results were reported in [99]. Thus the overall circuit MTBE is essentially equal to the mean time between sensitive bit upsets.

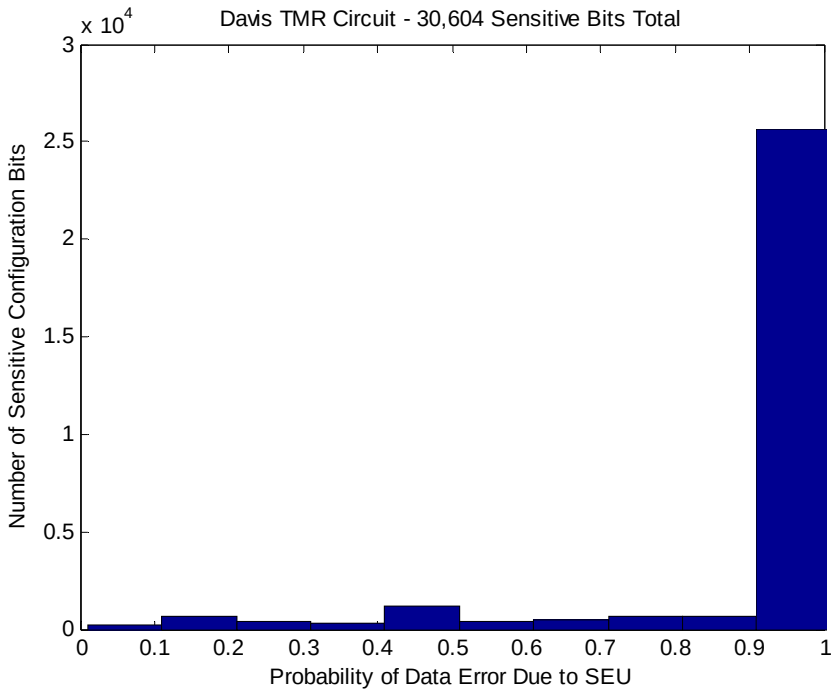


Figure 6.7 Histogram of Error Counts for Sensitive Bits in "Davis TMR" Circuit

The X1 control circuitry can perform more sophisticated error detection functions than simple correct vs. incorrect checks. In light of the fact that systems utilizing results from FPGA circuits can often tolerate a slight amount of imprecision, it is useful to investigate how often a circuit suffering SEUs is likely to provide grossly inaccurate data versus only slightly corrupted results. The RPR architecture is designed to ensure that output data is close to a correct result. Additional simulations were run with the RPR circuits to measure their probability of failing this goal. The data in Table 6.1 lists two values for the sensitive bit measurements for each RPR circuit. The first entries are based on a strict comparison for exact equality between X1 and X2 results. The second entries (listed in parentheses) are based on an error threshold that is only triggered when the X2 result differs from the X1 result in the 8 MSB positions. Since RPR is designed to ensure approximate equality, a true design failure only occurs when the output exceeds the upper/lower bounds. As seen in the table, this second failure criteria gives much lower sensitivity values. Although the "Davis RPR" data shows a tremendous decrease in sensitivity according to this second criteria (350 bits vs. 31,837 bits), it is important to realize that this data is skewed because it uses simple binary

counters for generating input values. The "Davis" circuits' counters are reset to all 0's after each artificial SEU injection and never count high enough to affect the 8 MSBs in the 1 msec before the next SEU is injected. Thus, many possible errors are not detected for the "Davis" circuit by using the approximate equality criteria. The simulations with the improved RPR circuits provide a fairer evaluation because they included LFSR random number generators.

It is also important to assess a fault tolerant design's ability to mask data errors. TMR and RPR are capable of hiding many internal data errors since the redundant module outputs are voted prior to being sent to output pins. Several of the test circuits provided an output flag indicating when the voter detected mismatches among the redundant modules. The data in Table 6.2 displays how many SEUs caused errors that were properly masked within X2, based on this output flag.

CORDIC Design Name (Virtex XQVR600cb228-4) (3,378,600 config bits)	Area (Slices)	# Masked SEU Errors	Fraction Masked Bits
Davis TMR Iterative	1326	90,821	2.69%
Davis RPR Iterative	540	29,376	0.87%
Improved 32-bit TMR Iterative	2541	193,409	5.72%
Improved 16-bit TMR Iterative	667	63,923	1.89%
Improved 32-bit RPR Iterative	933	32,729	0.97%
Improved 16-bit RPR Iterative	309	15,154	0.45%

Table 6.2 SEU Simulator Results for Masked Errors

Note that the TMR circuits have much larger values in Table 6.2 than in Table 6.1, whereas the RPR circuits have roughly equal numbers. This shows that the more comprehensive redundancy structure of TMR is able to detect and correct a greater percentage of SEUs. This is not surprising, however, since the RPR design is essentially blind to errors causing small numerical inaccuracies. In the RPR circuits tested here, 8-bit precision in the upper/lower bounds means many data errors are below the detection threshold.

The SEU simulator provides detailed data about the identity, timing and effect of each sensitive bit detected. However, the sheer scale of assessing the behavior

of over 3 million configuration bits makes the large datastreams produced by the simulator somewhat prone to error. Graphical displays of the data can be of great value for making qualitative assessments of each SEU simulation run. The results from an entire simulation can be displayed in a single visual image to check general behavior, or small segments of the data can be viewed in close-up plots to analyze finer details. Such graphical methods are commonly used in the field [85], [1]. A mapping of design sensitivity can be created by translating the bitstream addresses of each sensitive configuration bit to its physical device location. The left image in Figure 6.8 shows the sensitivity map created by plotting the locations of all 30,604 sensitive bits from the "Davis TMR" circuit.

The right image in Figure 6.8 shows the layout of the "Davis TMR" circuit, as displayed in Xilinx's "FPGA Editor" tool. This tool permits viewing and editing of a fully placed-and-routed circuit. FPGA resources that are actively utilized by the circuit (i.e., CLBs, muxes, clock buffers, I/O pins, etc.) are colored in the FPGA Editor display. Only SEUs affecting the active circuit elements should cause data errors.

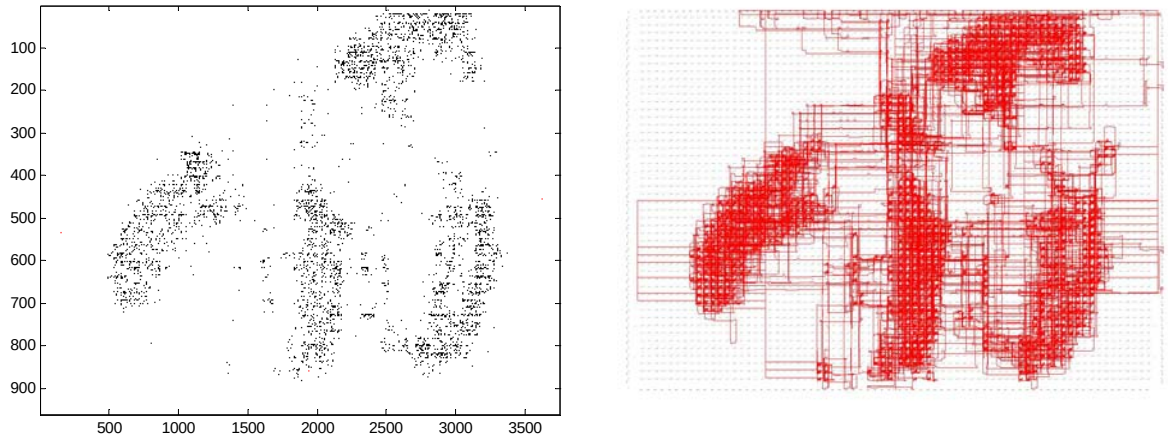


Figure 6.8 Detected Sensitive Bit Locations (left) vs. Circuit Layout (right) for "Davis TMR"

Note that the sensitive bits on the left are very well correlated with the actual circuit layout on the right. This provides assurance that the SEU simulator is properly injecting faults and detecting errors. For each simulation run, the SEU sensitivity map was compared to the circuit layout as a quality assurance check of the simulator data.

b. Discussion

As expected, the TMR circuits generally showed better SEU tolerance than the RPR circuits. This is not surprising, since the RPR design assumes that errors with small numerical consequence can be tolerated by downstream systems, and are therefore allowed to propagate. RPR's strength is in preventing errors in the most significant bits of the output data. As shown in Table 6.1, RPR demonstrated lower probability of suffering errors in the MSBs than across the entire output data vector. This supports the intuitive expectations from Chapter II. Comparing the number of errors in any bit position of the output data for the "improved" 32-bit and 16-bit CORDIC circuits, RPR appears to be roughly twice as sensitive as TMR (66,000 vs. 30,000 and 23,000 vs. 12,000). However, errors in the most significant 8 bits of the RPR output had similar frequency to errors anywhere in the TMR output. Using this evaluation criteria, RPR was better for the 32-bit version but worse for 16-bit version.. This difference is because the 8-bit redundant modules for RPR occupied a relatively larger fraction of the total circuit in the 16-bit case and therefore suffered a larger proportion of the total circuit-affecting SEUs.

The data also confirms that TMR is effective in detecting and correcting large numbers of SEUs. The TMR circuits show much higher counts of corrected errors (Table 6.2) than uncorrected errors (Table 6.1). For example, the improved 32-bit TMR design fixed more than six times as many errors as it was unable to fix. The basic TMR design includes three identical copies of an unprotected circuit. Therefore, one would expect the TMR voter to detect roughly triple the number of errors as appear in a single unprotected 32-bit circuit Indeed, the TMR circuit corrected 193,409 and the unprotected version produced 66,190 errors (a ratio of 2.9).

Although the general trends in the data follow many of the predictions made in Chapter II concerning the relative performance of TMR and RPR, it is quite surprising to see such high residual SEU sensitivity with both techniques in relation to the unprotected circuits. The rather high numbers listed in Table 6.1 prompted more careful examination of the SEU sensitivity maps. As expected, the voter and output logic regions of the circuits contributed to the SEU sensitivity of the designs. However, there were also many sensitive bits in the computation modules, which should be protected by the

redundant architecture. The quantity and location of these unexpected sensitive bits were reconfirmed through extensive retrials with full- and partial-device simulations, all of which supported the original data.

Several factors may contribute to the high sensitivity values for these circuits. First, as pointed out in Section 2, the circuit designs did not follow all of Xilinx's recommendations for implementing modular redundancy [68]. For example, Xilinx documentation describes using tri-state buffers instead of LUT logic for performing majority voting operations. The circuits tested here used a behavioral VHDL circuit description and allowed the vendor synthesis tools to optimize the design. Incorporating Xilinx's recommendations would require working more towards a structural VHDL design process to achieve maximum fault tolerance. A real operational system would clearly benefit from such an effort.

Another cause of the high sensitivity values may be the design synthesis software itself. There were several cases in which inconsistent or incorrect circuits were produced using Xilinx ISE 6.x series software. In some cases VHDL source code was tested in the ModelSim software to demonstrate proper behavior. However, after compiling and running the exact same source code on the actual FPGA, the results did not match the ModelSim results. Other problems were encountered trying to compile code using ISE 6.2 versus ISE 6.3. For example, version 6.2 for LINUX correctly synthesized VHDL code with a nested series of multiply and add operations whereas version 6.3 for Windows could not synthesize the same code. In another example, the VHDL code for a pipelined version of the CORDIC compiled correctly in 6.3 but not in 6.2. In light of these problems, and similar anecdotes from colleagues at other institutions, the CFTP group at NPS is planning to transition to a newer generation of the ISE software. Following this transition, it would be worthwhile to repeat some of these experiments to see if the high sensitivities persist.

Finally, some of these symptoms may be due to effects from "hidden half-latches" as described in [7]. These half-latches are used throughout the Virtex FPGAs for storing constant values of 1's and 0's as sources for muxes and other elements [43]. These memory elements are called "hidden" because their state is not reported through configuration readback and is set only upon initial device configuration (i.e., cannot be

fixed through partial reconfiguration). In addition to radiation-induced upsets of these half-latches, Graham explains how they can also be corrupted through SEU simulation [7]. Mitigation methods for half-latch issues exist, but were not used on the circuits tested in this research. Future work could involve testing these circuits after employing a half-latch removal method.

Further investigation into these unexpectedly high residual SEU sensitivities is an important area for follow-on research. More extensive radiation testing would be valuable for confirming these simulator results. It would also be useful to repeat the SEU simulations with TMR and RPR implementations that follow more closely the techniques proposed in Xilinx's application note on TMR methods [68]. For example, an RPR version of the 16-bit processor (with a 16-bit word for the full precision module and two 8-bit words for the approximate modules) could be built to fit within the existing CFTP configuration such that the results from all three modules could be output and voted off-chip. This would conform to Xilinx's recommendation of triplicating all input and output signals. However the CFTP system includes only 43 I/O pins connecting the X1 and X2 FPGAs, so full I/O replication is possible only for rather small data words.

While some of the results from these SEU simulations are not fully understood, the data do generally follow the predictions. An important contribution of this work is that a flexible and powerful SEU simulator is now available for testing various fault tolerant techniques. In particular, because the simulator is virtually identical to the CFTP flight hardware, it offers excellent reliability predictions for circuit designs destined for use during the CFTP space mission. The simulator can be used to compare the absolute and relative reliability of techniques such as TMR, RPR, selective redundancy, proprietary methods, and other concepts arising from the CFTP research program.

C. POWER SIMULATIONS

A primary advantage of an RPR approach over TMR is its reduced power consumption. To quantify this benefit, power estimates were made for the circuits tested

in Section B. Coupled with the SEU sensitivity data, these power estimates provide important information for comparing the effectiveness and efficiency of fault tolerant FPGA designs.

Power consumption can be estimated using detailed circuit models before a particular design is implemented. These power estimates require complicated models and, unfortunately, have limited accuracy due to uncertainties of various parameters [10]. Although the absolute accuracy of such power models is not perfect, they do offer valuable information about the relative power estimates for different circuits. Alternatively, the power consumption of a design can be assessed after implementation onto the FPGA by using high-fidelity power measurement equipment. The contributions from both static and dynamic power terms can be determined by measuring power consumption with and without clocking of the circuit. Since static power is independent of signal transitions, it is equal to the non-clocked power measurement. Dynamic power increases linearly with clock frequency. Therefore, a total power estimate is only valid for a specified clock frequency or when expressed as a function of clock frequency. Unfortunately, the existing power supply configuration on the CFTP hardware makes accurate power measurements impractical. A 28 V source supplies power to the entire CFTP package (CFTP board, ARM processor board and power conditioning board). This arrangement makes it very difficult to accurately isolate and measure the anticipated small (~10 mW) fluctuations on the X2 experiment FPGA. Therefore this research used computer modeling and simulations to estimate power.

1. Power Simulation Environment

Following the methodologies described by Rollins [9] and Tiwari [80], power simulations were performed using ModelTech's ModelSim software and Xilinx's XPower tool. Rollins found that ModelSim provided precise timing information for all signals on the chip and accurately captured the effects of signal transients. The accuracy of this process was verified by Rollins by precisely measuring power consumption on an actual hardware setup with Xilinx Virtex FPGAs. Though the CFTP hardware configuration was not amenable to direct power measurements, the power simulation software used by Rollins was available. Using a variety of circuits, Rollins demonstrated

that the simulation technique is fairly accurate. Rollins' largest discrepancy between actual and simulated power consumption was on the order of 50%, though most of the results agreed more closely. Though such a large difference may be a cause for concern when seeking high precision estimates, it is less significant when comparing circuits that differ by 200% or more. It was expected that power consumption of the RPR and TMR designs would differ by much more than 50%. Therefore, it was determined that the uncertainties in the models would be relatively insignificant and power simulations would provide sufficient accuracy for this research. Nonetheless, an important topic for further investigation is the verification of these simulation results with actual power measurements on the CFTP hardware. Such work would require modifications to the CFTP circuit board(s) and/or power supply setup.

Several conditions are required to make accurate power estimates. These include 1) a fully defined circuit description, 2) realistic signal toggle rates for each node, and 3) precise capacitance values for all signal lines. The first condition means that the circuit under investigation must be completely mapped and placed-and-routed for the particular FPGA device being used. Circuits described in schematic or high-level languages such as VHDL must be compiled into FPGA-specific formats completely describing the locations and interconnections between design elements. The second condition indicates that a probabilistic or simulation-based estimate is needed to provide activity rates used in computing the dynamic power component. The third condition is also critical in computing dynamic power. While static power is a constant value for a particular FPGA device (see Chapter III), the second and third conditions allow for accurate determination of dynamic power consumption.

Figure 6.9 shows the interaction of the various software tools used in the power simulation process. Typically, the design is expressed as a set of VHDL source code compiled through a commercial FPGA synthesis package. The most commonly used synthesis software includes Xilinx's ISE suite and Synplicity's Synplify products. Several more steps are required to create the final circuit design that can be loaded onto the target FPGA. The mapping and place-and-route steps produce a Native Circuit Description file (*.ncd) that can be viewed and/or edited in the FPGA Editor tool. Processing this file through BitGen creates a Bitstream file (*.bit) that can be loaded onto

the device. The NCD file can also be processed through a Xilinx utility program called "NetGen" that outputs a "flattened" VHDL version of the circuit and a System Data File (*.sdf) containing the timing information needed for high-fidelity circuit simulations.

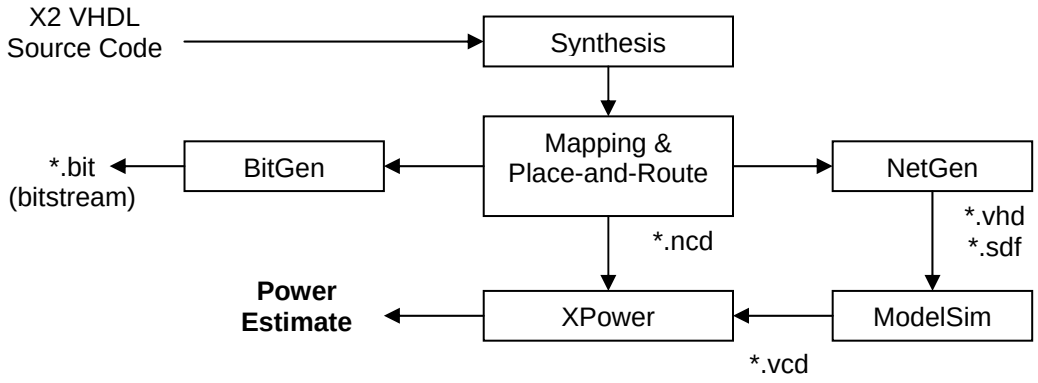


Figure 6.9 Power Simulation Process Flowchart

ModelSim is then used to simulate the entire circuit at high time resolution. By combining the information in the VHDL and SDF files, ModelSim predicts the sequence of signal transitions at every node in the design. The accuracy of the simulation is controlled through a selectable time resolution setting. In general, shorter timescales provide more accurate results, but create large output files. A timescale of 10 psec was chosen for the CORDIC simulations to ensure the proper simulation of glitching events. The output from the ModelSim simulation is a Value Change Dump file (*.vcd) characterizing the frequency at which each signal node toggles.

Finally, XPower combines the VCD and NCD files to calculate the total power consumption for the design. XPower is essentially a proprietary database of capacitance values for each FPGA element. Using the formulas for dynamic power presented in Chapter III, XPower uses this database and the signal transition data from ModelSim to output several power consumption reports. For this work, the average power consumption was the most appropriate metric for comparing different circuits.

The most important variables in the ModelSim runs were the frequency of the input clock, the length of the simulation run and the timesteps requested of the simulator. To match the CFTP hardware configuration, the data in Table 6.3-Table 6.4 was generated with a 50 MHz input clock. Each run spanned 100 microsec of simulation

time. This yielded results relatively quickly, prevented file sizes from growing too large, and ensured that the random number generator went through enough unique values to provide good statistical coverage. To investigate the possible dependence on time resolution, the unprotected 32-bit iterative circuit was tested with several different timestep settings. As shown in Table 6.3, power estimates increased only slightly as the time resolution was reduced below 100 psec. Thus a 10 psec setting was used for the remaining runs.

Timestep Increment (psec)	1	10	100	1000
Total Power Estimate (mW)	184	184	182	172

Table 6.3 Power Estimates for Unprotected 32-bit Iterative CORDIC Circuit

2. Test Circuits

The same circuits used for SEU simulations were run through the power simulation process described above. First, the two "Davis" circuits used for radiation testing were evaluated. Those 32-bit CORDIC circuits used simple binary counters and had the most primitive voter designs of the fault-tolerant circuits tested. Next, several 32-bit CORDIC variants were tested, including an unprotected design, a TMR design and an RPR design. Data from these circuits were intended to show the relative power usage of TMR and RPR solutions. Likewise, three different implementations of a 16-bit CORDIC circuit were tested. In addition to allowing comparison of TMR and RPR, the 16-bit circuits also provide insight into the relationship between circuit size, complexity and power. As described in Section B, the circuits built after the UC Davis radiation testing had various improvements, most notably the pseudo-random number generators (LFSRs) that create more realistic signal toggle behavior than the simple counters.

In addition, power usage was evaluated for two pipelined CORDIC circuits that were not tested in the SEU simulator. These circuits were tested to support assumptions made in earlier chapters about the potential power savings of a pipelined architecture. Following the procedures described in the previous section, additional circuits could easily be analyzed for power consumption to determine the most efficient designs for achieving fault tolerant FPGA solutions.

3. Results

Table 6.4 presents the results gathered from the power simulations. The rightmost columns in the table give the dynamic and total power consumption estimated from the XPower tool. The output from XPower gives separate estimates for both static and dynamic power. As explained in Chapter III, static power is a fixed quantity for a given FPGA device, regardless of the circuit implemented on the chip. For the XQVR600 device, XPower reports static power of 32.16 mW. Therefore, the difference between the two right columns is simply this constant static power component. Static power accounted for between 1% and 30% of the total power consumption for the circuits tested.

Dynamic power, on the other hand, depends on the circuit design loaded on the FPGA and the input values to that circuit. All of the CORDIC designs were self-contained. The only inputs were a clock and single reset line. This made running simulations in ModelSim easier, as only a small sequence of test stimulus code was needed.

CORDIC Design Name (Virtex XQVR600cb228-4)	Area (Slices)	Dynamic Power (mW)	Total Power (mW)
Davis TMR Iterative	1326	453	485
Davis RPR Iterative	540	142	174
Improved 32-bit TMR Iterative	2541	477	510
Improved 32-bit RPR Iterative	933	172	204
Unprotected 32-bit Iterative	843	152	184
Improved 16-bit TMR Iterative	667	147	179
Improved 16-bit RPR Iterative	309	77	109
Unprotected 16-bit Iterative	238	73	105
Unprotected 32-bit Pipeline	2058	2137	2169
Unprotected 16-bit Pipeline	519	438	470

Table 6.4 Power Estimates from XPower

As expected, all of the TMR designs use significantly more power than the RPR and unprotected versions of each circuit, supporting the primary motivation for pursuing

the RPR concept as a means of conserving power. Focusing on the dynamic power data, the 32-bit TMR circuits used 2.8-3.2 times as much power as RPR, while the ratio between the TMR and RPR 16-bit circuits was 1.9. In fact, the RPR circuits used only marginally more power than the unprotected CORDIC designs, with a power overhead of between 5% and 13%. The small 8-bit lookup tables used for the upper/lower bounds calculations occupy only a small portion of the FPGA and the RPR voter is fairly compact. Thus it is not surprising that RPR shows terrific power performance.

Careful examination of Table 6.4 reveals that, while the 32-bit TMR circuit uses about triple the power of the 32-bit unprotected circuit, the 16-bit circuits do not match expectations. The TMR designs contain three copies of the unprotected circuit plus voter logic, thus one expects that TMR should use at least three times the power. Further investigation revealed that the surprisingly low power value for the 16-bit TMR circuit is due to the lack of triplicated clock and output signals. The TMR and RPR designs built for this research utilize single-clock circuits with on-chip voting, in part due to I/O limitations on the CFTP hardware. The clock and output signals account for a large fraction of the dynamic power consumption in the relatively small 16-bit unprotected design, which occupies only 3% of the total FPGA. To demonstrate this effect, two additional circuits were simulated with triplicated clock and output pins. These circuits cannot be run on the actual CFTP hardware, but can be tested in ModelSim and XPower. Table 6.5 shows that when the clock network and output data is fully triplicated (thereby obviating the voter logic), the TMR design does indeed consume three times as much dynamic power. Note that the unprotected circuit's power value here is higher than that in Table 6.4 because the clock dividers in the original circuits had to be eliminated (the Virtex FPGA allows a maximum of 4 clock networks) and so the 50 MHz input clocks were driving all circuit components.

CORDIC Design Name (Virtex XQVR600cb228-4)	Area (Slices)	Dynamic Power (mW)	Total Power (mW)
Modified 16-bit TMR Iterative (triplicated clock and outputs)	730	1083	1115
Modified 16-bit Iterative (with clock divider removed)	231	314	347

Table 6.5 Power Estimates for Modified 16-bit CORDIC Circuits

As explained in [9], dynamic power consumption can also be measured as the slope of total power as a function of frequency. By running the same circuit at various clock frequencies, the dynamic power can be isolated from the total power. While XPower provides this information directly, it is more difficult to measure using actual hardware. Though this research did not involve measuring power on actual FPGA devices, such work would be useful for future research. Having actual measurements and computer simulations for various clock frequencies would aid in validating the simulations. Table 6.6 shows the frequency-dependent power estimates for two of the test circuits. Again, the TMR circuit requires more than twice as much power as the RPR design.

CORDIC Design Name (Virtex XQVR600cb228-4)	Total Power (mW)			Slope (mW / MHz)
	25 MHz	50 MHz	100 MHz	
Improved 32-bit TMR Iterative	334	510	855	6.95
Improved 32-bit RPR Iterative	139	204	331	2.56

Table 6.6 Dynamic Power Inferred from Power Gradient

Another way of interpreting the data from Table 6.4 is to determine the correlation between circuit size and power consumption. As mentioned in earlier chapters, smaller circuits generally consume less power. Figure 6.10 shows a scatter plot for the 8 iterative CORDIC circuits tested. The circuits roughly follow the trend line, though the trend line was pulled upward by the “Davis TMR” data point. In fact, both the “Davis TMR” and “Davis RPR” circuits use roughly the same amount of power as the corresponding “Improved” circuits but are physically much smaller. This is because the

XST synthesis software performed poorly when compiling the VHDL version of the 32-bit CORDIC module (see Appendix A). By excluding that data point, the lower trend line shows very good correlation between circuit size and power consumption. This result indicates that circuits with similar function and structure can be expected to have an approximately linear power-to-area relationship.

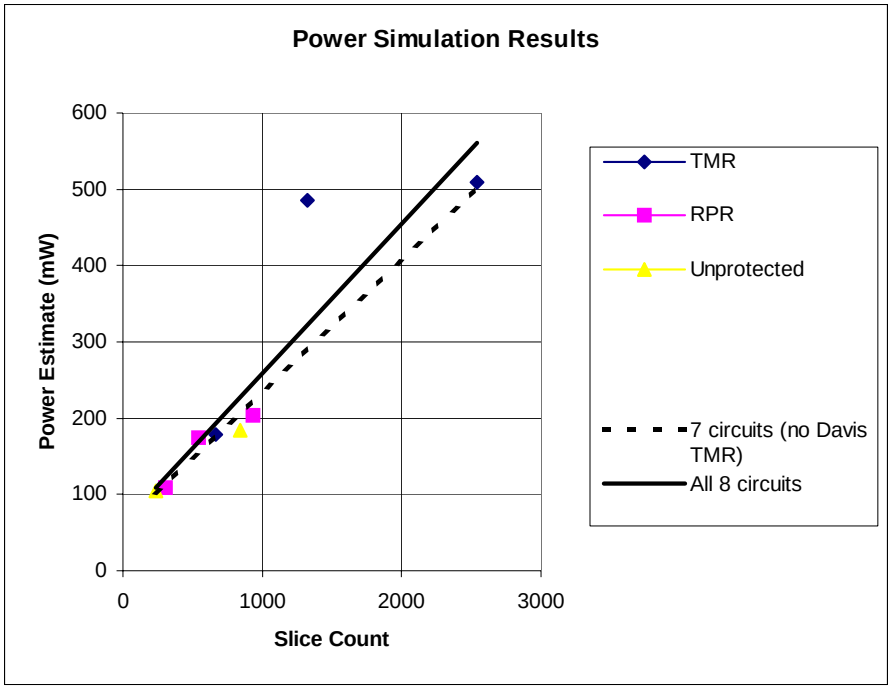


Figure 6.10 Scatter Plot of Power Consumption vs. Circuit Size

Though the pipelined circuits were designed to compute the same function as the iterative designs, their structure was entirely different. Therefore the pipelined circuits did not follow the same power-area function as the other circuits. Table 6.4 shows that the pipelined circuits use significantly more power than the iterative versions. However, the pipelined CORDIC design is much more energy efficient on a per-calculation basis because of its greater throughput. For example, the throughput of the 32-bit pipelined circuit is 37 times that of the iterative circuit (see Appendix A), though it only uses 12 times as much power. Similarly, Rollins [100] demonstrated that pipelined multiplier circuits can reduce both energy-per-calculation and overall power consumption since pipelining reduces glitching power (see Chapter III). If circuit area is not as significant a

constraint as power, there are options for using a pipeline approach to achieve lower power. For example, one might use a large pipelined circuit with lower clock frequency to achieve similar throughput to an iterative design. Alternatively, one might run the pipelined circuit at full speed for short bursts of time and pause between processing sets of data to achieve lower average power consumption. The tools and processes used in this section are valuable for assessing these kinds of design alternatives without relying upon hardware measurements.

D. SUMMARY

This chapter demonstrates that RPR is a viable option for achieving fault tolerance with minimal power cost. Fault injection analysis with the unique CFTP SEU simulator shows that RPR is effective at reducing the probability of suffering SEU-induced faults in the most significant bits of a circuit's output. Though the TMR circuits are roughly twice as effective at eliminating SEU sensitivity across all of the output data bits, RPR provides comparable SEU reduction across those bits protected by the error bounds calculations. The TMR designs also require significantly more power. Data from power simulations verify that TMR uses roughly twice the power of RPR. Thus for applications where slight inaccuracy is acceptable or power is a significant constraint, the RPR architecture is superior to the typical TMR approach.

Aside from the data presented in this chapter, this work has developed a set of tools and processes for effectively comparing different design options. These tools allow for more informed decisions regarding the trade-offs involved in choosing a fault-tolerant architecture. Data from SEU and power simulations feed directly into the total performance metric described in Chapter IV. This permits the evaluation of many possible design alternatives before building hardware or relying on expensive radiation chamber testing.

THIS PAGE INTENTIONALLY LEFT BLANK

VII. RADIATION TESTING

A. OVERVIEW

1. Purpose

Given the many uncertainties involved in predicting the behavior of complicated electronic devices like FPGAs, it is important to have real-world data to support the conclusions from simulations and modeling. In this context, "real-world" means subjecting actual FPGA circuits to radiation levels sufficient to generate SEUs as they would occur while on-orbit. Radiation experiments were conducted in August and November of 2005 at Crocker Nuclear Laboratory on the campus of UC Davis. These tests were intended to 1) validate the SEU simulation environment, 2) verify the operation of the CFTP fault detection/correction techniques and 3) estimate the actual on-orbit response of the CFTP experiment. Although extensive radiation testing has been performed on the same Virtex-II XC2V6000 device used in some of our experiments, to our knowledge this was the first live proton testing with the Virtex XQVR600 device. It was expected that these experiments would corroborate results reported elsewhere in the literature for similar Xilinx FPGAs, but the primary concern in this work was the performance of the fault-tolerant architectures and techniques developed under the CFTP project.

The main focus of this dissertation is predicting the performance of the RPR architecture in comparison to the more common TMR approach. Estimating system reliability requires several assumptions. In particular, computer system reliability in space depends on radiation conditions, the susceptibility of digital circuits to that radiation, and the complex interaction of signals within a circuit. The space radiation environment has been studied extensively and detailed models are available for general use. However, the remaining two factors are specific to a given technology and design. The radiation response of complicated devices, such as FPGAs, is difficult to predict. Real-world data gathered by exposing actual hardware to realistic radiation sources provides important validation of the simulation techniques described in Chapter VI.

Several other research groups have conducted radiation testing to characterize the SEU response of FPGAs. For example, data errors caused by configuration-bit faults,

which account for about 98% of all errors [99], are much more common than those due to flip-flop upsets. Configuration bits in the I/O blocks, lookup tables and routing elements have similar susceptibility to SEUs, though the LUT bits are the most susceptible [44]. Another important observation is that not all configuration-memory upsets lead to output data errors [42], [44], [85].

The CFTP test results reconfirm many of these conclusions. More importantly, these results validate the SEU sensitivity measurements reported in Chapter VI. Finally, this new data can be combined with other environmental and device data to make accurate predictions of on-orbit performance for the two CFTP experiments soon to be launched.

2. Test Equipment, Setup, and Designs

Radiation testing was performed at the Crocker Nuclear Laboratory using the cyclotron high-energy proton source as part of pre-launch testing of the NPS Configurable Fault Tolerant Processor (CFTP). The monoenergetic 63.3 MeV proton beam was tuned to provide a nearly uniform 4 cm square irradiation pattern on the devices-under-test. The radiation flux was controlled during testing to yield the desired SEU rate and total dose. Limited data was collected during a short test run in Aug 2005, whereas the Nov 2005 testing produced a substantial volume of data.

Two hardware configurations were tested at the Crocker facility. The first configuration included two identical Xilinx Virtex FPGAs, one running the experimental circuits ("experiment FPGA") and one commanding the experiment FPGA and controlling data flow in/out of the board ("control FPGA"), as shown in Figure 7.1 below. This configuration is labeled "C1" and corresponds to the CFTP flight experiment to be launched on the NPSat-1 and MidSTAR-1 satellites in late-2006. The second configuration uses the same Virtex control FPGA, but has a more advanced Virtex-II device as the experiment FPGA. This is labeled "C2" and is a rough prototype design for possible future CFTP missions. Both configurations use a 50 MHz system clock, which is frequency divided within each FPGA into 25 MHz and slower clocks, as required by the various test circuits.

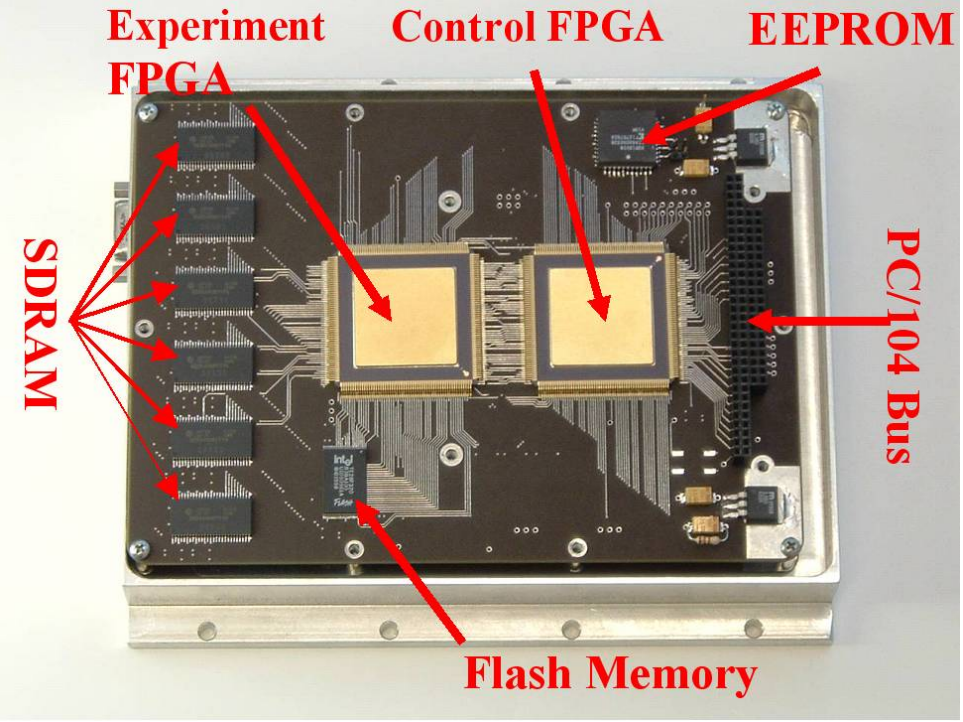


Figure 7.1 Hardware Configuration "C1"

Several designs were tested on each hardware configuration. Three shift register designs were implemented in both the C1 and C2 configurations. These densely-packed shift register designs were expected to exhibit the highest sensitivity to SEUs since they had very high utilization of FPGA logic resources. Two CORDIC designs, corresponding to the TMR and RPR designs examined in Chapter VI, were implemented on both hardware configurations. Finally, a pipelined MIPS-like microprocessor design with distributed TMR error detection and voting was tested on the C2 hardware (this was not tested on the C1 hardware because it exceeded the logic capacity of the XQVR600 device). Table 7.1 lists the names used for each of these circuits, where "…c1" and "…c2" refer to the same circuit implemented on the different hardware configurations. Section B presents results for each hardware/circuit combination. More detailed descriptions of the test equipment, setup and test circuits can be found in [98].

Design Name	Description
sr_SRL_c1 (c2)	Parallel shift registers w/ SRL16 macro
sr_SRL+1_c1 (c2)	Parallel shift registers w/ SRL16 and flip-flops
sr_noSRL_c1 (c2)	Parallel shift registers w/ flip-flops only
cordic_GOLD_c1 (c2)	32-bit CORDIC w/ TMR
cordic_APPROX_c1 (c2)	32-bit CORDIC w/ RPR
PIX_c2	MIPS-like microprocessor w/ distributed TMR

Table 7.1 Names and Descriptions of Test Circuits

3. Test Procedure

Figure 7.2 outlines the basic procedure followed during radiation testing. These procedures are similar to those described in other FPGA radiation experiments [44], [99]. It was important to control the rate at which SEUs occurred in order to isolate which configuration bit upsets were responsible for observable data errors. In these experiments an SEU rate of one every 30 seconds was chosen to match some earlier diagnostic procedures for the CFTP experiment. This rate is considerably slower than the SEU time interval of between 1 and 5 seconds in [44] and the interval of 1 second in [99]. Subsequent enhancements to the CFTP setup permit higher data rates and future testing could easily support SEU rates of one per second or higher.

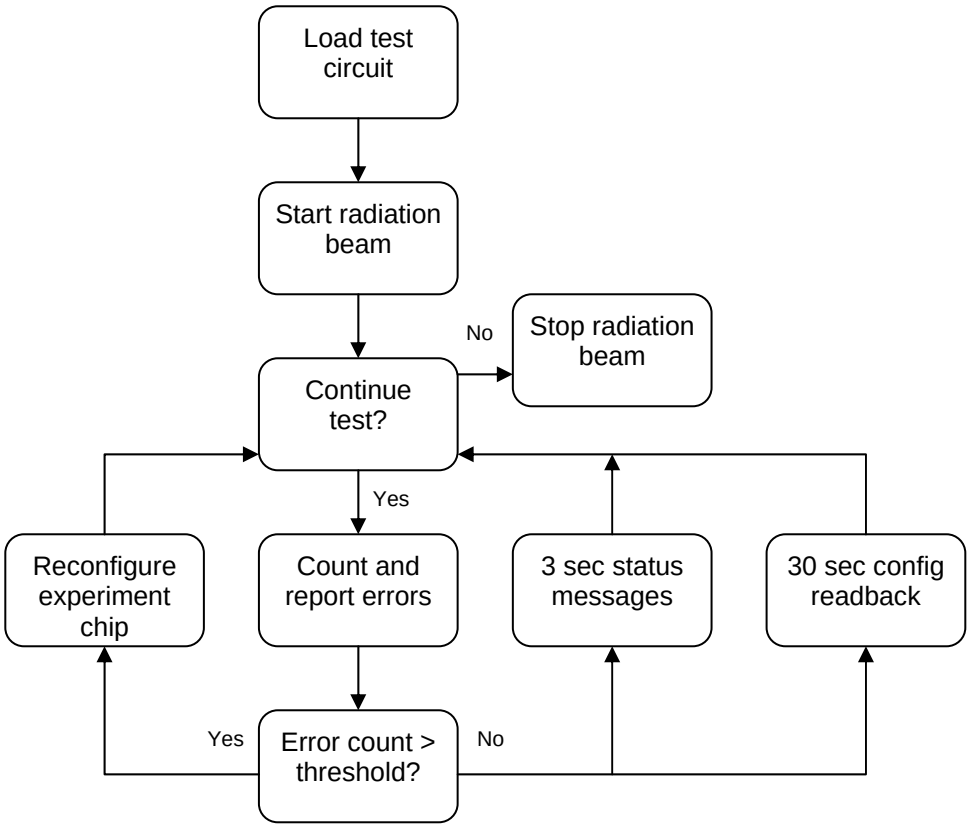


Figure 7.2 Radiation Beam Test Procedure

Most of the test time for each circuit was spent looping between the "Count and report errors" and "3 sec status messages" steps while on-board counters automatically generated new input data vectors. Every 3 seconds an output message was generated that reported the state of the input, output and error count values from the experiment and control FPGAs. As long as the radiation-induced SEUs did not lead to observable data errors at the circuit output, no reconfiguration or device reset was necessary. SEUs were allowed to accumulate until data errors were observed. Every 30 seconds a complete configuration readback was performed and all accumulated SEUs reported. Later analysis of the datastream identified which SEUs were preexisting and which were new during each readback.

SEUs affecting flip-flops have temporary effects, may only cause a small number of data errors, and do not necessitate a device reconfiguration. On the other hand, SEUs affecting configuration bits persist indefinitely, generate many data errors and can only be corrected through reconfiguration. Therefore, various error counters were

implemented on the control FPGA to keep track of the number of data errors produced by the experiment FPGA. These counters automatically triggered a reconfiguration of the experiment FPGA when the error counter(s) reached a predetermined threshold, indicating an error-producing configuration fault had occurred. In addition, the test team was able to manually reconfigure and reset the experiment, as needed.

B. RESULTS

A 1 krad maximum radiation dose was set for each hardware configuration. The C1 configuration sustained a total radiation dose of 730 rads during its 104 minutes of beam time, while the C2 configuration experienced 620 rads in 76 minutes of exposure. These total dose values were well below the 1 krad goal for this testing. Although the devices tested are predicted to survive a total dose of at least 100 krad, a much more conservative limit was placed on this testing to avoid damaging the sole CFTP spaceflight prototype hardware. Furthermore, a low dose was desired for this test campaign so that additional future testing could be performed on the same devices. Table 7.2 summarizes the data collected during this testing. Runs 1 through 6 were conducted with the C1 configuration and generated a total of 390 configuration SEUs. Runs 7 through 16 used the C2 setup and generated 2,258 configuration SEUs. The following sections describe the results in more detail.

Design and Run #	Total # SEUs	Re-configs	Beam time (sec)	Sec per SEU	Multi-bit upsets
sr_SRL_c1_run1	27	5	489	18.1	1
sr_SRL+1_c1_run2	35	11	329	9.4	2
cordic_GOLD_c1_run3	79	1	1,346	17.0	0
cordic_APPROX_c1_run4	106	0	1,915	18.1	0
sr_noSRL_c1_run5	26	1	421	16.2	0
sr_SRL+1_c1_run6	117	26	1,755	15.0	4
sr_SRL+1_c2_run7	47	9	86	1.8	1
sr_SRL+1_c2_run8	414	52	758	1.8	8
cordic_GOLD_c2_run9	319	5	458	1.4	5
cordic_APPROX_c2_run10	419	0	669	1.6	5
PIX_c2_run11	681	8	1,374	2.0	11
sr_SRL_c2_run13	172	18	302	1.8	0
sr_noSRL_c2_run14	172	57	330	1.9	5
sr_noSRL_c2_run15	27	16	143	5.3	0
sr_SRL+1_c2_run16	75	17	446	5.9	2

Table 7.2 Summary of Radiation Test Results

1. Fluence-to-SEU and Cross Section

A fundamental measure of a device's radiation susceptibility is the amount of radiation required to cause a fault and/or error. This information permits the prediction of upset rates in different orbital regimes. In this research, the main concern is the sensitivity of FPGA configuration bits since flip-flop upsets are much less common. Configuration-bit upsets are classified as faults, as they do not necessarily lead to actual data errors. In this chapter the term SEU refers specifically to upsets affecting configuration bits, since the tests were designed to only detect this type of fault. As explained earlier, these bits comprise the vast majority of memory elements on the FPGAs, accounting for over 99% of faults during radiation testing and on-orbit operations [99].

The column "Sec per SEU" in Table 7.2 shows a fairly consistent SEU rate within each hardware configuration, with a few notable outliers. A more precise measurement

of radiation susceptibility can be gained by looking at the actual fluence levels, as given in Table 7.3. Fluence, as measured in protons per cm^2, depends on the intensity of the radiation beam and the exposure duration. The cyclotron was tuned to nearly the same flux levels for test runs 1 through 14. However, for test runs 15 and 16 the cyclotron was set to a lower beam current to reduce the SEU rate on the more sensitive Virtex-II device. This explains the apparent anomaly in the bottom two rows of Table 7.2. The bottom two rows of Table 7.3 show "Fluence-to-SEU" values consistent with the other C2 test runs, as expected. The C1 results in Table 7.3 compare favorably with data presented in [99], where the authors found a fluence-to-SEU of between 9.8 and 13 p^+/cm^2 for three test circuits on a Virtex 1000 FPGA.

Design and Run #	Total # SEUs	Fluence (p^+/cm^2) x 10^8	Fluence-to-SEU (p^+/cm^2) x 10^6	Sec per SEU
sr_SRL_c1_run1	27	4.08	15.1	16.9
sr_SRL+1_c1_run2	35	2.91	8.3	8.2
cordic_GOLD_c1_run3	79	11.9	15.1	17.0
cordic_APPROX_c1_run4	106	16.8	15.8	18.1
sr_noSRL_c1_run5	26	3.70	14.2	16.2
sr_SRL+1_c1_run6	117	15.5	13.2	13.9
sr_SRL+1_c2_run7	47	0.71	1.5	1.8
sr_SRL+1_c2_run8	414	8.63	2.1	1.8
cordic_GOLD_c2_run9	319	3.88	1.2	1.4
cordic_APPROX_c2_run10	419	5.61	1.3	1.6
PIX_c2_run11	681	12.1	1.8	2.0
sr_SRL_c2_run13	172	2.70	1.6	1.8
sr_noSRL_c2_run14	172	2.97	1.7	1.9
sr_noSRL_c2_run15	27	0.35	1.3	5.3
sr_SRL+1_c2_run16	75	1.06	1.4	5.9

Table 7.3 Fluence-to-Upset by Test Circuit

As expected, the C2 experiment FPGA was more sensitive than the C1 device. C2's more advanced Virtex-II device is based on 0.15 micron technology compared to

0.22 micron technology in the C1 Virtex device. These smaller dimensions permit greater logic density. "Equivalent gate" count is a common method for comparing FPGA capacity. Xilinx lists the equivalent gate counts for the XQVR600 and XC2V6000 devices tested as 600,000 and 6,000,000 gates, respectively. Based on this ratio, one would expect the Virtex-II device to be roughly 10 times more sensitive since both devices have similar semiconductor die sizes. Summing the "Total # SEUs" and "Fluence" values from all runs in Table 7.3, the fluence-to-upset ratios are 14.1×10^6 for C1 and 1.69×10^6 for C2. This indicates that C2 is more than 8 times as sensitive as C1, which corresponds fairly well with the rough estimate of 10 based simply on equivalent gate counts.

Another metric commonly used in radiation studies is cross section (see Chapter IV). Data from Table 7.3 can be combined with the size of the configuration memory on each device to estimate configuration bit cross sections. The number of configuration memory cells on each device is approximately equal to the size of the configuration bitstream files. The C1 device's bitstream contains 3,607,968 bits and the C2 device has 21,849,504 bits. This translates into proton cross section values of $2.0*10^{-14}$ and $2.7*10^{-14}$ cm^2 per configuration bit. Previously published data on another Virtex device gives a proton cross section of $2.2*10^{-14}$ cm^2 per configuration bit [88]. This matches very well with the data collected for the CFTP experiments.

Finally, it should be noted that the anomalously-high SEU sensitivity observed on test run 2 is not fully understood. Nothing in the test logs indicates a problem with the radiation source or device under test. Test run 6 on C1 and runs 7 & 16 on C2 used the same "SRL+1" circuit design, but showed sensitivity values similar to the other circuits. Thus, it appears that run 2 was a statistical outlier or that some unknown factor affected its results.

2. Variability of SEUs and Bit Sensitivity

While the preceding analysis shows totaled values for each data run, it is interesting to note the considerable variability of SEU rates within each run. The random nature of the proton-induced SEU process causes a distribution in the rate of SEU events. This can be seen by plotting SEUs as they are detected throughout the tests. The

following two figures show the SEU time history for data runs with the RPR version of the 32-bit CORDIC processor. Figure 7.3 shows the SEU profile from run 4, which used the C1 hardware. Note that the SEU rate varies between 0 and 6 SEUs during each 30 second readback cycle. This highlights the need to collect large enough data sets to average out this variability.

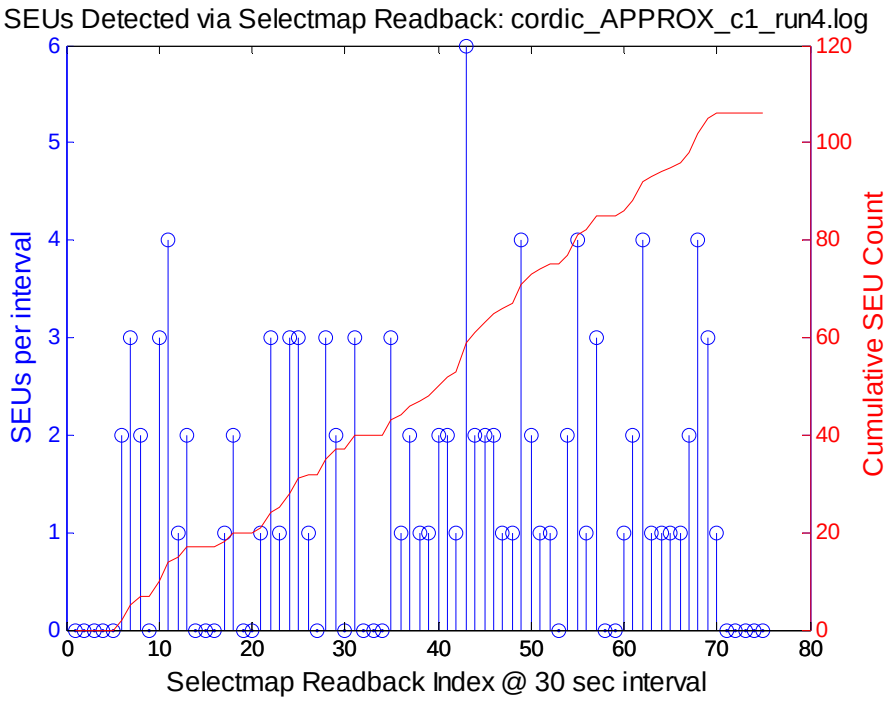


Figure 7.3 SEU Profile for Run 4 (C1 Hardware)

Figure 7.4 shows the SEU time history from run 10, which used the C2 hardware. This run also demonstrates the fluctuating rate at which SEUs appear on the device. This rate ranges from 7 to 28 SEUs per readback interval, with an average value of 18.

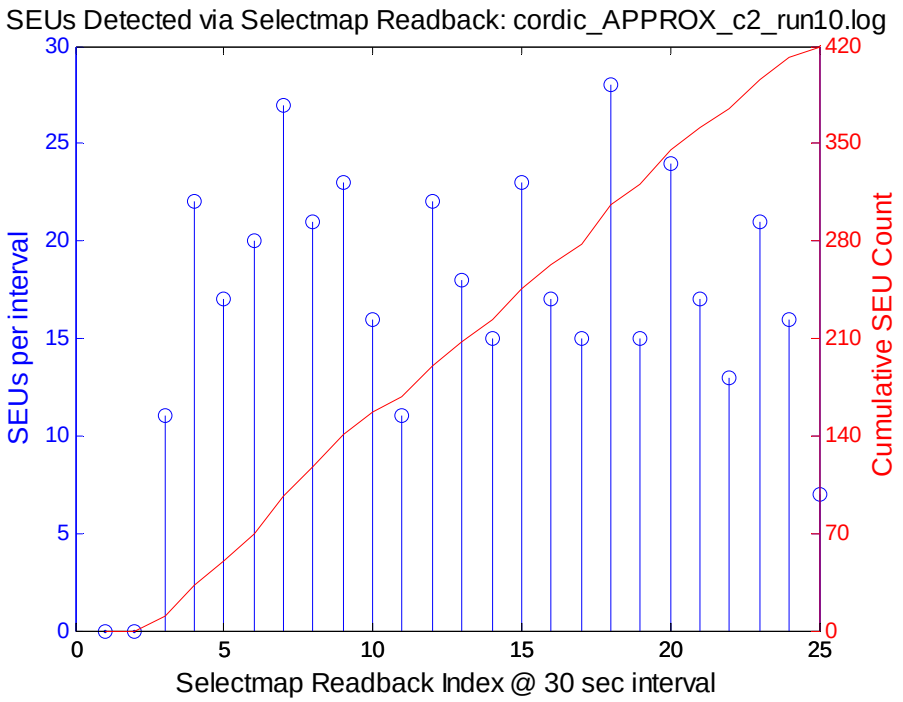


Figure 7.4 SEU Profile for Run 10 (C2 Hardware)

Another important conclusion from this testing is that most configuration-bit upsets do not cause data corruption. The term "sensitive bits" is described in [99] and is often used to classify those particular configuration bits that, when upset, can lead to errors in output data. The column labeled "Reconfigs" in Table 7.2 indicates how often configuration-bit SEUs caused observable data errors and, consequently, triggered a device reconfiguration. In some of the shift register test circuits, it is possible that flip-flop upsets could also trigger reconfiguration if they occur near the beginning of the shift register "train," but this is estimated to account for less than 5% of the reconfigurations. Comparing the number of reconfigurations to the SEU counts yields an average sensitivity fraction of 11% for C1 and 8% for C2. Thus about 1 in every 10 configuration SEUs is likely to cause data errors.

A large reason for this phenomenon is that most designs do not fully utilize the FPGA resources. Unused portions of the device do not contribute to the sensitive-bit count. Nonetheless, even designs that heavily utilize the FPGA show much smaller sensitive-bit populations than might be expected. Other radiation tests have also shown this behavior. For example, data from ion testing in [44] shows error-to-SEU ratios of

between 1:6 and 1:783, depending on which circuit and ion species were tested. Another proton study reports a sensitivity fraction of between 5% and 15% [99], which is very similar to results from CFTP testing.

3. Polarity of Bit Flips

Another interesting question is whether there is any preferential polarity with which SEUs occur. An assumption made in the fault injection simulator and associated reliability estimates is that 0-to-1 and 1-to-0 bit flips are equally likely. As seen in the far right column of Table 7.4, the C1 circuits appear to have roughly equal ratios of 0-to-1 and 1-to-0 flips, whereas the C2 circuits have a dramatically higher number of 0-to-1 flips.

To better understand this discrepancy, one must look at the ratio between 0 and 1 values in the fault-free configuration bitstream for each circuit design. If there were no preferential polarity of bit flips, then the observed ratio of upsets should match the ratio of bitstream 0 and 1 values. Table 7.4 shows the bitstream count and observed SEU polarity ratios for each circuit design.

Design and Run #	Bitstream			SEU Polarity		
	"0" Bits	"1" Bits	Ratio	0→1	1→0	Ratio
sr_SRL_c1_run1	2,564,645	1,043,323	2.46	16	13	1.23
sr_SRL+1_c1_run2	2,625,121	981,847	2.67	15	25	0.60
cordic_GOLD_c1_run3	2,547,015	1,060,953	2.40	53	26	2.04
cordic_APPROX_c1_run4	2,534,019	1,073,949	2.36	52	54	0.96
sr_noSRL_c1_run5	2,562,171	1,045,797	2.45	13	13	1.00
sr_SRL+1_c1_run6	-	-	2.67	76	50	1.52
C1 Totals & Average Ratios	12,832,971	5,205,869	**2.47**	225	181	**1.24**
sr_SRL+1_c2_run7	18,979,250	2,870,254	6.61	45	3	15.0
sr_SRL+1_c2_run8	-	-	6.61	366	53	6.91
cordic_GOLD_c2_run9	21,758,566	90,938	239	320	4	80.0
cordic_APPROX_c2_run10	21,810,370	39,134	557	423	2	212
PIX_c2_run11	21,244,996	604,508	35.1	663	18	36.8
sr_SRL_c2_run13	19,104,432	2,745,072	6.96	164	8	20.5
sr_noSRL_c2_run14	19,548,310	2,301,194	8.50	160	17	9.41
sr_noSRL_c2_run15	-	-	8.50	25	2	12.5
sr_SRL+1_c2_run16	-	-	6.61	67	9	7.44
C2 Totals & Average Ratios	122,445,924	8,651,100	**14.2**	2,233	116	**19.3**

Table 7.4 Comparison of Observed SEU Polarity and Bitstream Values

These results are somewhat surprising because they show a large difference between the expected SEU polarity ratios and those observed in actual radiation testing. The C1 circuits all had actual ratios less than predicted from the bitstream analysis, with a nearly equal rate of 0-to-1 and 1-to-0 bit flips. Several factors can help explain this discrepancy. The Xilinx bitstream data files all include numerous "padding" bits throughout the bitstream. These pad bits ensure the proper alignment and flushing of various registers during device configuration. By default, pad bits are set to zero. Since the pad bits do not correspond to actual configuration memory cells, they cause an upward skew to the "bitstream ratio" values. More than 6% of the bits in each configuration frame are pad data, thereby accounting for some of this discrepancy. In addition, there are large BlockRAM elements on the Virtex FPGA that were not used in the C1 test circuits for these experiments. The configuration readback program for C1

was not set up to detect SEUs in these portions of the device. The bitstream values for these unused BlockRAM regions are set to zero, accounting for over 3% of the bitstream. Ignoring these two populations of predominately zero values reduces the average predicted SEU polarity ratio for C1 from 2.47 to 2.24. There is still a significant difference between this estimate and the observed ratio of 1.24, thus it appears that 1-to-0 SEUs have higher likelihood than 0-to-1 SEUs on the Virtex device.

Data from C2 showed greater variability among the test circuits and, in general, an opposite trend to that seen with C1. The actual SEUs observed favored the 0-to-1 polarity. Although the BlockRAM bit upsets were detectable in the C2 setup, the issue of pad bits causing a bias towards higher zero-counts in the bitstream files also applies to the C2. However, this would exacerbate the discrepancy between the predicted and observed values. Thus, the Virtex-II device seems more susceptible to 0-to-1 bit flips.

Further investigation into these phenomena is warranted. One way of gaining a better understanding is through examining the transistor properties of each device. A similar study with an SEU-hardened Atmel FPGA revealed that 0-to-1 upsets were roughly 50 times more likely than 1-to-0 upsets [101]. The explanation for this strong bias was the particular layout of NMOS and PMOS in the configuration memory cells. An analysis of the Xilinx devices may yield similar explanations, and based on these test results one might expect to find that the Virtex and Virtex-II parts use different memory structures. However, such analysis would require detailed information about the internal design of Virtex devices, which is not made publicly available by Xilinx.

These test results have highlighted that FPGAs may be more susceptible to certain polarities of configuration-bit upsets. The standard industry practice for calculating orbital upset rates assumes a single sensitivity value (e.g., fluence-to-upset or cross section), regardless of polarity. If additional testing and/or analysis confirms that there are indeed preferred bit-flip orientations, an intriguing possibility for improving FPGA reliability would involve developing bitstreams with higher percentages of the less susceptible bit polarity.

4. Multiple Bit Upsets (MBUs)

Although the discussion up to this point has assumed that each SEU corresponds to a single bit flip, it is possible for single particle events to upset multiple bits. This phenomenon, known as multiple bit upset (MBU), is well-known [17] and recent results have been published describing MBU probabilities on FPGA devices [102]. Multiple bit upsets occur when a single energetic particle either directly or indirectly generates sufficient excess charge density in a localized region to affect more than one memory cell. MBUs are more common with ion radiation than proton radiation, as heavy ions transfer much more energy into the semiconductor and thereby affect a larger region of the device.

MBUs are becoming more common as new FPGAs are developed with denser and more sensitive logic elements. Proton and heavy ion radiation testing has demonstrated the progression in MBU frequency over several generations of Xilinx's Virtex devices. As a percentage of all proton-induced SEU events, MBUs accounted for 0.04% on the Virtex, 1% on the Virtex-2, and 3% on the Virtex-4 FPGAs. With ion testing, 7% of all Virtex and 35% of all Virtex-2 single events involved the upset of multiple bits [102].

As expected, a small number of MBUs were observed during testing of the CFTP devices. Total MBU counts for each test run are listed in Table 7.2, where each MBU event is counted as a single SEU. Averaged over all the runs on the Virtex board, they accounted for 1.8% of the configuration upsets. On the Virtex-2 board they totaled 1.6% of all SEUs. While the Virtex-2 results correlate fairly well with the results in [102], the Virtex data here shows a surprisingly high frequency of MBUs. One possible explanation for this is that several of the test circuits (labeled in Table 7.2 as "sr_SRL…") utilize the Virtex "SRL" feature, which converts some of the FPGA's 4-input lookup tables into 16-bit shift registers. All of the MBUs detected in the Virtex runs involved these SRLs, suggesting this may be the cause of the high MBU rates. Additional testing is required to resolve whether this anomaly is due to the particular circuit being tested or the physical behavior of the Virtex device.

The CFTP SEU simulator discussed in Chapter VI only injects one fault at a time and does not simulate MBUs. This is also true for other SEU simulation systems [8], [99]. MBUs are neglected in these simulations primarily because they are rare, and

therefore have a small affect on design sensitivity and reliability calculations. However, MBUs are more important when using newer devices or considering orbits with significant populations of heavy ions. MBUs could be included in the CFTP simulator with minor modifications.

C. VALIDATION OF SIMULATIONS

While the preceding section demonstrates that the CFTP system accurately detects and reports faults in a real radiation environment, the most important result from radiation testing is the validation of the SEU simulator of Chapter VI. Following the methodology in [99], the CFTP radiation test results for the C1 configuration were compared against predictions from the SEU simulator. These results show that the simulator accurately simulates radiation-induced SEUs. This validation provides confidence in the reliability assessments made in Chapter VI, in particular the relative performance of RPR and TMR.

Two methods were used in this validation process. The first method involved manually injecting into the simulator specific faults observed during radiation testing and verifying that the data outputs responded in the same manner. Since the radiation flux was controlled to ensure only a small number of SEUs occurred during each 30 sec configuration readback cycle, the exact bits causing persistent data errors could be easily isolated. These bits were then toggled in the controlled lab environment to verify whether or not they caused observable data errors in the radiation environment. Due to the tedium of this process, only a small fraction of the radiation-induced faults were recreated in this manner. Nonetheless, the CFTP response in the simulator matched that seen during testing in all trials. Table 7.5 shows verification data for the first 3 SEUs observed during radiation testing of the TMR version of the CORDIC design. The columns under the heading “Data Error?” show equivalence between the radiation testing and artificial fault injection results. The other columns identify the specific configuration bits for the manual fault injection mode, as detailed in [103].

Design and Run #	Byte Location	Read vs. Expected	Maj. Addr.	Tile (Row,Col) Bit (Row,Col)	Data Error?	
					Sim.	Rad.
cordic_GOLD_c1_run3	01d31d	0x82 vs 0x02	21	(49,53) (08,27)	No	No
	027593	0x38 vs 0x18	28	(27,26) (06,38)	No	No
	02e026	0x02 vs 0x00	33	(46,59) (12,25)	Yes	Yes

Table 7.5 Example of Manually Verifying SEU Effects

The second method involved exhaustive simulator testing of every bitstream value using the automatic mode. This automatic mode is the main operational mode for the simulator and was used in generating the design sensitivity results in Chapter VI. Design sensitivity data for both the TMR and RPR versions of the 32-bit CORDIC processor were compared against the 185 configuration upsets observed during testing. Comparing results from upsetting these same 185 bits validated the automated mode of the SEU simulator. Table 7.2 shows that only 1 of these upsets led to an actual data error, which was verified as shown in Table 7.5. The single sensitive bit (located within byte 0x02e026 of the TMR CORDIC design) causes data errors in both the simulator and radiation data. Likewise, the remaining 184 bits are all found to be non-sensitive in the simulator and radiation data.

Figure 7.5 shows a close-up view of a portion of the SEU sensitivity map generated from the simulator. The black squares in the figure represent individual configuration bits that, when artificially upset in the simulator, lead to data errors. The remaining configuration bits are non-sensitive and appear in the map as white squares. Radiation-induced SEUs that occur in the white regions are not expected to lead to data errors. The red square (circled) represents a bit that experienced an SEU during radiation testing, but did not cause any data errors. This red square coincides with a white non-sensitive bit, thus corroborating the simulator data declaring this a non-sensitive bit. This comparison method was used to verify that that all 184 non-sensitive bits from radiation testing correspond with non-sensitive bits as determined by simulations.

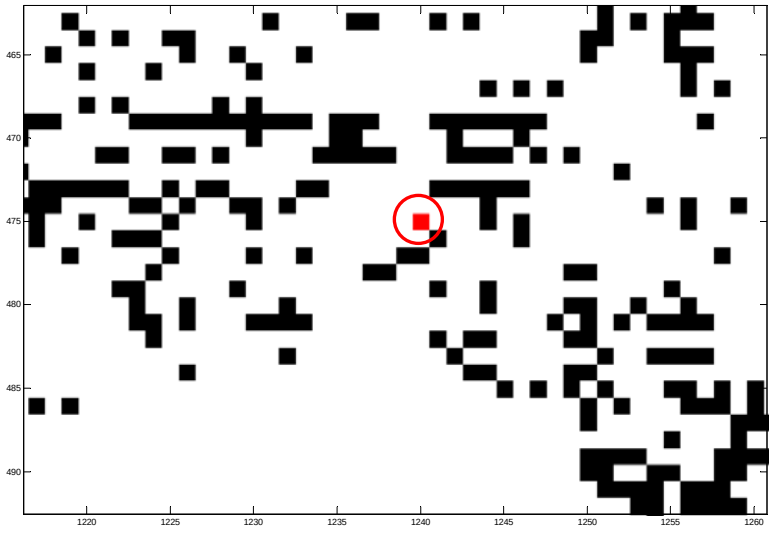


Figure 7.5 Locations of Sensitive Bits (Simulator, black) and SEUs (Radiation, red)

Likewise, the single sensitive bit detected during radiation testing was replicated in the simulator. Figure 7.6 shows the region of the FPGA in which the error-producing SEU appears in the Davis data. Again, the black squares show error-producing bits found in the simulator and the circled red square shows a non-error producing bit that was upset in radiation testing. The important feature shown in this figure is the green square, surrounded by a diamond, which is the location of the error-producing SEU found during run #3 of Davis testing. The green color indicates that both the simulator and radiation data show this bit to be error-producing.

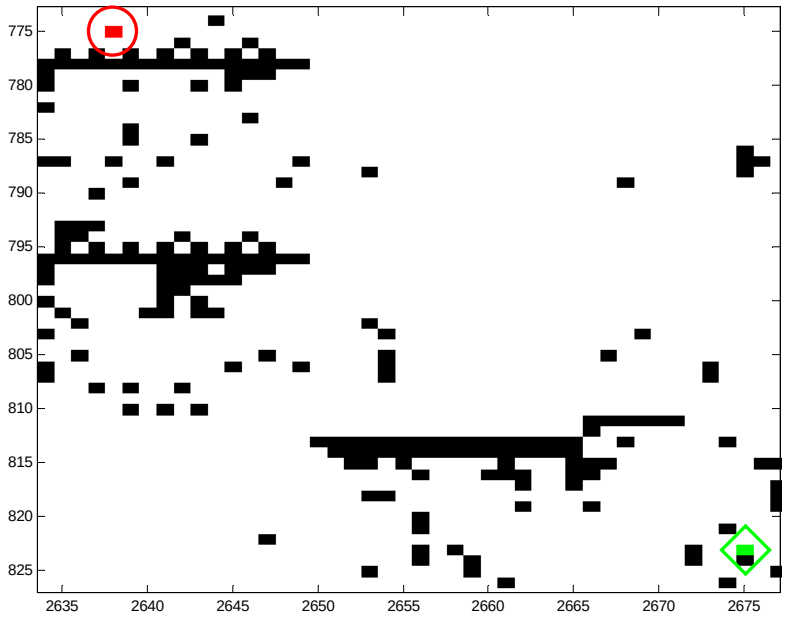


Figure 7.6 Correlation of Sensitive Bit (green) from Simulator and Radiation

Run #4 of the Davis testing identified 106 SEUs, none of which triggered a reconfiguration. Simulator data indicates that two of these 106 bits are actually capable of causing data errors. However, the simulator shows that these errors occur in the least significant positions of the output data. Thus, the 8-bit RPR upper/lower bounds checking would not detect such errors and trigger a reconfiguration. The radiation-induced SEUs affecting these two particular bits occurred during the 32nd and 35th readback cycles. Analysis of the run #4 data file shows that the LSBs in X2's output did, in fact, differ from the X1 solution, starting in readback cycle 32. This implies that the first of these SEUs was responsible. The effect of the second SEU is unclear because the first SEU was not corrected (no reconfiguration occurred) and therefore continued to affect the data output. Throughout the remainder of run #4 the least significant bits of the 32-bit X2 output had random errors.

The version of the SEU simulator used in this dissertation was customized to work only with the CFTP flight system. Unfortunately, the simulator is not directly compatible with the Virtex-II board. Therefore, radiation data with the C2 configuration was not used in this validation effort. As listed in Table 7.2, the CORDIC circuits on the

C2 hardware experienced 5 data-corrupting bit upsets and a total of 738 SEUs. All 5 of the data-corrupting SEUs occurred on the "GOLD," or TMR, version. This data would be of great benefit if the SEU simulator were expanded to work with the Virtex-II board.

Although very few sensitive bits were encountered in radiation testing, both sensitive and non-sensitive bit SEUs provide valuable confirmation of the accuracy of the SEU simulator. Furthermore, since the results from Chapter VI show a low percentage of sensitive bits, it isn't surprising that only 3 out of 185 SEUs caused detectable errors for the CORDIC designs on C1. The SEU simulator measured these circuits to have bit sensitivity percentages of 1% to 3%. Therefore, one would expect to see less than 6 sensitive bits out of a population of 185.

D. ON-ORBIT RELIABILITY

The primary motivation for conducting ground-based radiation testing and SEU simulations is to estimate a system's reliability in its operational environment. Combining test and/or simulation data with models of the space radiation environment provides useful predictions of how often a particular device is likely to malfunction. This information is critical for determining whether system-level reliability requirements can be met with a particular design solution.

On-orbit reliability has long been a concern to the spacecraft computing community, and interest in the radiation tolerance of FPGAs is rapidly increasing. In a space experiment similar to the CFTP experiment, the Australian FedSat satellite was launched in Dec 2002 with a high-performance computing experiment package including a Xilinx XQR4062XL FPGA. The designers predicted an SEU interval of between 9.7 minutes and 110 hours for a 680 km sun-synchronous orbit [3]. The large range in this estimate is due to the variability in space radiation levels caused by solar activity. It is important to note that the FedSat FPGA is an older generation than the Virtex parts used in the CFTP experiment (62,000 vs. 600,000 equivalent gates). If CFTP flew in a similar orbit to FedSat, it would be expected to have roughly 10 times the SEU rate, or between 2 and 1,400 SEUs per day.

Another FPGA space computing program is currently underway at Los Alamos National Lab with the Cibola Flight Experiment satellite [16]. This satellite will be launched in late-2006 on the same STP-1 mission as the CFTP experiments and uses a total of 9 Xilinx Virtex XQVR1000 devices. In the planned 560 km 35º inclination orbit, each Virtex FPGA is expected to see between 3 and 100 SEUs per day, depending on solar flare activity [1]. The FPGAs on CFTP are the same generation as the Cibola devices, but are somewhat smaller (600,000 vs. 1,000,000 equivalent gates), and should therefore suffer roughly 60% as many SEUs, or between 2 and 60 per day.

Accurate estimation of on-orbit performance requires a thorough understanding of the space radiation environment. The spacecraft's orbit determines the quantity, composition and variability of ionizing radiation the spacecraft will encounter. In low earth orbit (LEO), trapped high-energy protons are the dominant source of SEUs, though their population varies as the earth's magnetosphere fluctuates in response to solar activity. Furthermore, protons in the region known as the South Atlantic Anomaly (SAA) are responsible for the vast majority of SEUs in LEO, even though most LEO spacecraft spend less than 15% of their orbits in the SAA region. At geosynchronous altitude, SEUs are predominately caused by cosmic rays and solar flare particles. The radiation environment at high altitudes varies widely due to solar flare activity [17].

Models of the complex and variable space radiation environment include AP-8, SPACERAD, CHIME, JPL and CREME. The expected SEU rates quoted earlier for the FedSat and Cibola spacecraft were based on these computer models. The European Space Agency hosts a website called SPace ENVironment Information System (SPENVIS) that provides access to the most commonly used radiation models for making detailed predictions of the space environment. These models were used in this research to predict upset rates of CFTP in its planned 560 km orbit. Considering only the trapped proton population, the orbit-averaged flux of sufficiently energetic protons (>20 MeV) varies between 60 and 200 per cm^2 per second, depending on solar conditions and which model is used. Based on the data from Section B above, these radiation levels should produce 0.37-1.2 SEU/day in the Virtex device and 3-10 SEU/day in the Virtex-II device. Using the CREME96 models and some slightly different assumptions, Coudeyras

predicted a Virtex upset rate of 0.2 SEU/day and Virtex-II rate of 1.8 SEU/day [98]. Table 7.6 compares the SEU rate estimates calculated from the CFTP test data with those extrapolated from similar studies [1], [3].

Method of Calculation	SEU per day (LEO orbit)	
	C1 (Virtex)	C2 (Virtex-II)
CFTP radiation test data + SPENVIS models	0.37 – 1.2	3 – 10
CFTP radiation test data + CREME96 models	0.2	1.8
FedSat data (scaled from XQR4062 results)	2 – 1,400	20 – 14,000
Cibola data (scaled from XQVR1000 results)	2 – 60	20 – 600

Table 7.6 Summary of Radiation Test Results

The numbers based on the CFTP-SPENVIS and CFTP-CREME96 data are considerably lower than the values extrapolated from studies on similar devices. Because the cross sections measured in the CFTP tests are very similar to other published test results, this discrepancy must be due to differences in the assumed proton flux levels. For example, the values produced from the SPENVIS tools may be somewhat low. They estimate between $5*10^6$ and $1.7*10^7$ p^+/cm^2/day, whereas graphs in [17] show between 10^7 and 10^8 p^+/cm^2/day with energy greater than 30 MeV without solar flare enhancement.

Differences in how the models predict strong solar flare events could account for the discrepancy in the upper bounds. There is disagreement in the literature about the intensity of “normal” and “anomalously large” solar flares, which is reflected in the different radiation models. Two additional factors may explain the discrepancy in the lower bounds. First, the calculations performed here discount any affect of heavy ions (cosmic rays), as most researchers consider them negligible in LEO. The FedSat and Cibola data may include the contribution of these energetic ions. Second, the FedSat and Cibola data do not report the proton energy threshold used in their studies. A threshold lower than the 20 MeV used here would increase the estimated SEU rates since the proton population decreases exponentially as a function of energy level. Some of the researchers involved in the Cibola study indicate in a separate report that they typically use a threshold of 10 MeV.

While determining SEU rates is an important part of estimating reliability, the SEU rates alone are not sufficient to compare the overall fault tolerance of different designs. From a system-level perspective, the primary concern is operational reliability, or the probability that the system provides correct results. The functional reliability of an FPGA circuit can be predicted by combining estimated SEU rates with design sensitivity measures from a fault simulator like that in Chapter VI. For example, if only 10% of the configuration bits in a particular design are capable of causing data errors, the actual failure rate would be 10% of the SEU rate. Fault-tolerant FPGA designs seek to minimize the fraction of SEUs that lead to errors.

Comparison of various fault tolerance methods must be based on the functional reliability of each approach. Fault injection simulations provide the design sensitivity data for competing design alternatives. This data can then be used to calculate the reliability metric incorporated into the Total Performance Metric of Chapter IV. Together, the simulation and TPM tools provide a objective method for evaluating the overall efficiency of various fault-tolerant designs.

An important result from this chapter is the validation of the SEU simulator from Chapter VI. Analysis of nearly 200 proton-induced SEUs affecting two CORDIC test circuits shows that the C1 hardware behaves identically in the simulator and the ground-based radiation test environment. Hence, it appears that the simulator accurately represents the response of the CFTP hardware to real space radiation conditions. The successful performance of the configuration readback and reconfiguration mechanisms in both the simulator and radiation testing gives confidence that these procedures will function effectively during CFTP flight experiments. Once the CFTP experiments are launched, these conclusions can be verified by SEU data in the real orbital radiation environment. In addition, analysis of over 2,500 SEUs reveals proton cross section values for Virtex and Virtex-II FPGAs of $2.0*10^{-14}$ and $2.7*10^{-14}$ cm^2 per configuration bit, respectively, which is comparable to the published value of $2.2*10^{-14}$ cm^2. Finally, using this data to estimate on-orbit SEU rates indicates that in its low-earth orbit CFTP will experience only one to two upsets per day.

THIS PAGE INTENTIONALLY LEFT BLANK

VIII. PRACTICAL RPR IMPLEMENTATION ISSUES

A. OVERVIEW

This chapter addresses some practical concerns about applying RPR to real-world problems. Building on the discussion from Chapter II, the first section addresses whether a particular algorithm can be converted into approximated versions to provide the upper and lower bounds calculations for the RPR architecture. The discussion focuses on the practical aspects of creating useful and accurate error bounds calculations. The second part of the chapter presents a real-world application to demonstrate the appropriate use of RPR. An image compression algorithm provides a case study to investigate the effect of inexact results in an RPR system due to SEUs. This section describes a methodology to estimate such effects and improve the system design process. As the case study demonstrates, it is quite acceptable in many applications to occasionally produce lower precision results, especially considering the power and area savings of RPR over a TMR approach.

B. WORKING WITH UPPER AND LOWER BOUNDS ESTIMATES

Chapter II acknowledges that although many applications and algorithms are amenable to the RPR approach, some are not. Computational problems are classified as either Class A or Class B depending on whether RPR is an appropriate method of gaining fault tolerance. Figure 2.12 describes the steps for determining if a problem is Class A or B. Step 3 of this process asks whether approximate solutions can be formulated. Chapter II approaches this question by considering how functions affect the relationships between input and output "clusters." This chapter takes a more practical approach to explore how typical numerical functions can be approximated and implemented in FPGAs.

1. Properties of Numerical Functions Amenable to Approximation

A function must meet several conditions in order to apply the approximation techniques described below. First, the function must be well-behaved. That is to say that it is single-valued, continuous over the (possibly restricted) domain of input values, and

has a continuous first derivative. A common function that violates the single-valued restriction is the square root. If both positive and negative solutions are permissible, two equally correct solutions are possible. Thus, depending on the calculation method(s), there may be ambiguity when developing approximations to compare against full-precision solutions. For example, a Newton-Raphson method, which converges towards a final answer over a series of iterations, might produce positive or negative solutions depending on the initial guess and computational precision. The sign function, defined as +1 for inputs greater than 0 and as –1 for inputs less than 0, and the step function are other common functions that do not meet the criteria of well-behaved. Due to discontinuity at the origin, they may evaluate to different solutions depending on numerical precision and rounding.

A second, and more restrictive, condition that is relaxed in subsequent sections is that the function should be monotonically increasing or decreasing. This ensures consistency with regard to the method of calculating the upper and lower bounds. Figure 8.1 shows how the relationship between the precise solution and the high/low estimates changes depending on whether the function has a positive or negative slope. In regions with a positive slope, the lower bound can be calculated based on a rounded-down version of the precise input value. In regions with negative slope, processing this rounded-down input value yields an upper bound.

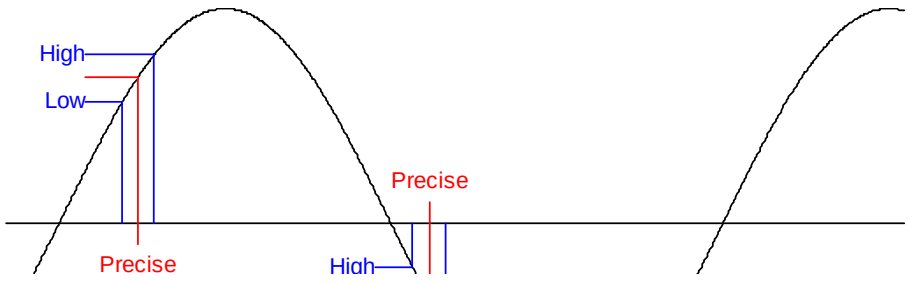


Figure 8.1 Upper/Lower Bounds for Sine Function

For periodic functions, such as the sine function in Figure 8.1, one can define a restricted range of allowable input values over which the function is constantly increasing or constantly decreasing. Values outside this range could be "mapped" into this range through appropriate shifting by some multiple and/or fraction of the period. For the sine function, this involves addition/subtraction by integral numbers of π. For aperiodic non-monotonic functions, one needs to either restrict input values to a monotonic region of the function or incorporate special techniques for dealing with regions where the slope changes sign, as discussed below.

It is important to maintain a consistent convention for calculating the lower/upper bounds and comparing them with the precise solution. The voter in an RPR architecture requires an upper bound that is always greater than or equal to the precise solution. Likewise, the voter's lower bound input must be less than or equal to the precise solution for all input values. Otherwise numerical comparisons made by the voter will be non-deterministic and the design will be greatly complicated.

Ensuring the proper relationship between the precise, upper, and lower solutions is especially challenging near stationary points, where the function's slope changes from positive to negative, or vice versa. The following section discusses different techniques for calculating the upper/lower bounds using an abbreviated version of the precise input value. One common method is to calculate the lower bound from a rounded down input value and calculate the upper bound from a rounded up input. A simple voting convention assumes that the output computed from the smaller input value is the lower bound and the output from the larger input value is the upper bound.

Figure 8.2 shows the difficulty with this approach when the input value is near a stationary point, which is marked by the green arrow. The left frame of the figure shows the desired situation, with the upper and lower values properly bounding the precise solution. The right frame shows what happens after the function crosses the stationary point. In this situation the "low" estimate is actually larger than the "high" estimate, since the function's slope has changed from positive to negative. This invalidates the voting mechanism, which requires the precise solution to be *larger* than the low estimate and *smaller* than the high estimate. When the low and high input values straddle the stationary point, the situation can be complicated even more. The center frame shows an

example where the precise input produces an output value larger than that calculated from either the low or high inputs. Thus, even in a fault free condition, the voter detects an error since the precise solution does not fall within the bounds.

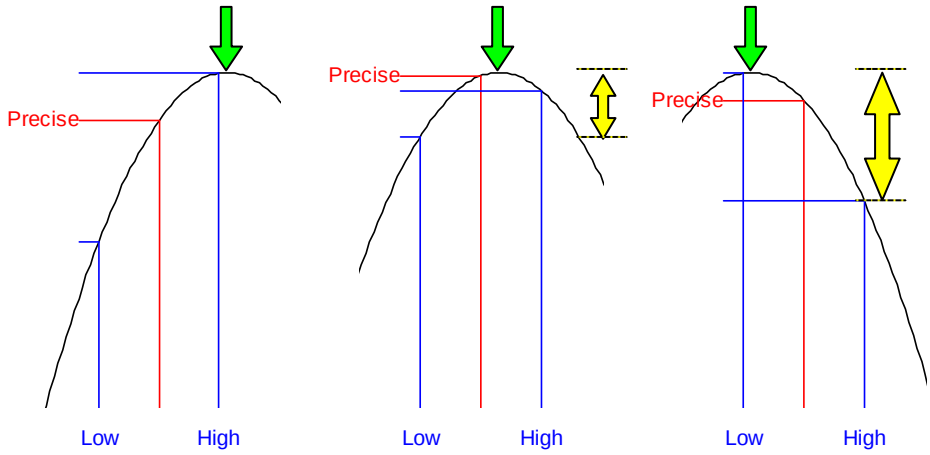


Figure 8.2 Upper/Lower Bounds in Vicinity of Stationary Points

Also shown in the right two frames of Figure 8.2 are the proper error bounds (yellow double-headed arrows) surrounding the precise solutions. In the middle frame, the output computed from the "high" input falls below the precise output value. The proper upper bound must be equal to or greater than the maximum possible precise output value over the low-to-high range. In the right frame, the upper and lower bounds surround the precise value, but their relationship to the "low" and "high" inputs are swapped relative to the left frame. A voter designed to work with upper/lower bounds following the convention of the left frame (low input→lower bound and high input→upper bound) would not properly resolve the situation shown on the right.

2. Lookup Tables versus Direct Calculation

Some of the problems described in the previous section regarding non-monotonic functions can be alleviated by calculating the upper/lower bounds with lookup tables. In fact, in many cases a lookup table approach is preferred. Assuming the precision required for the approximate solutions is reasonable, lookup tables are relatively small

and more power efficient than methods that perform mathematical calculations. Lookup tables are easy to implement in modern FPGAs and can be customized to any particular function.

The general architecture for calculating upper and lower bounds in parallel with a precise solution is shown in Figure 8.3. The input vector is divided into two sections. The most significant (leftmost) bits are supplied to all three functional blocks while the least significant bits are supplied only to the precise calculation. Though the variables *m* and *n* will vary between applications, the values 8 and 24 are used here to correspond with the 32-bit full-precision and 8-bit approximate CORDIC calculations used elsewhere in this dissertation.

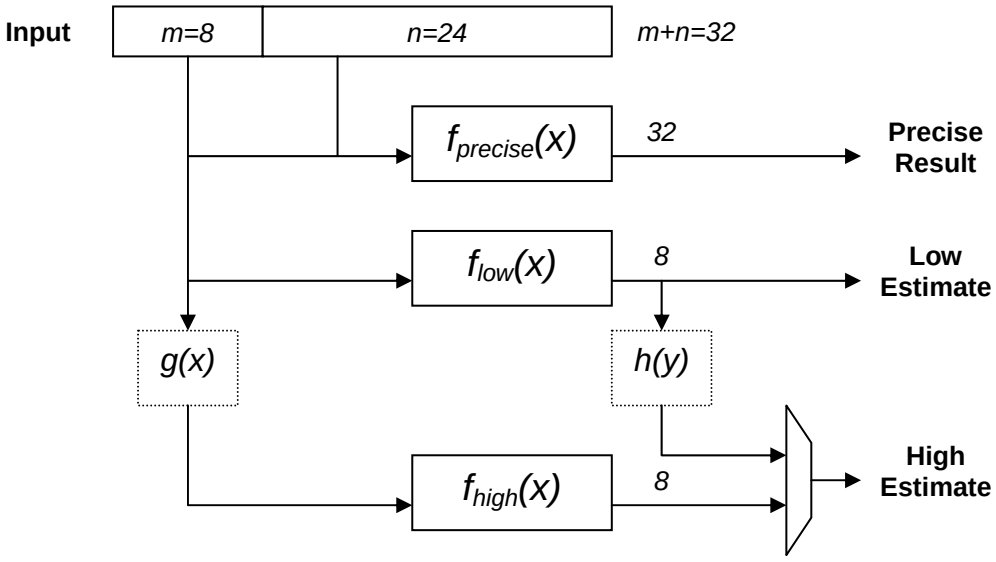


Figure 8.3 Architecture for Calculating Precise and Upper/Lower Bounds Solutions

A fundamental premise leading to the layout shown in the figure is that, for monotonic functions, the precise solution can be bounded by independent calculations using values that bound the precise input number. Note in the figure that the "low" and "high" estimates may use different versions of the input vector MSB block. The input to $f_{low}(x)$ is essentially a truncated version of the precise input value. For standard binary number systems (unsigned, two's complement, one's complement, etc.), truncation represents a rounding down of the input. The optional block *g(x)* often simply

increments the input (though it may be necessary to check for overflow conditions or other special cases). Alternatively, this block can be eliminated if the function $f_{high}(x)$ is designed to operate directly on the truncated input values. A lookup table for $f_{high}(x)$, for example, can be programmed to accept the truncated values, making $g(x)$unnecessary.

The three main functional blocks shown in the figure can be completely unique solutions to the numerical function being solved. For example, if the numerical function is addition, the precise solution might involve a fast carry look-ahead structure while the approximate modules may use a simpler ripple-carry approach. The RPR circuits in this dissertation compute sine and cosine using the CORDIC method for the precise solution and lookup tables for the upper/lower bounds.

The outputs from the low and high estimates are shown as m-bit values, corresponding to the precision that can normally be achieved given an m-bit input. However, in some situations these outputs can be expressed more precisely. For example, lookup tables can provide arbitrarily high precision outputs by generating values through high-resolution offline computations. This is useful for maintaining tight error bounds, especially in regions where the function has a small slope.

In some cases, it is possible to replace the block $f_{high}(x)$ with a simpler conversion of the low estimate. The operation $h(y)$ applies prior knowledge of the expected relationship between the low and high estimates to convert the output from $f_{low}(x)$. For example, the high estimate can be calculated by adding the maximum difference between low/high bounds to the low estimate. In this way the operations $g(x)$ and $f_{high}(x)$ can be replaced with a simple adder in $h(y)$. A drawback to this approach is that it reduces SEU fault tolerance since faults in the $f_{low}(x)$ block lead to inaccurate estimates of both the lower and upper bounds. However, the function $h(y)$ can be useful in conjunction with separate $f_{low}(x)$ and $f_{high}(x)$ blocks by verifying that their results are properly spaced.

Although lookup tables are easy to use, care must be taken to ensure they correctly represent the upper and lower bounds of the function throughout the entire input space. To generate the contents of the lower bound lookup table, the full precision solution for all 2^m possible truncated input values must be calculated. Assuming the function has a positive slope, these high-precision solutions computed can then be truncated (i.e., rounded down) to yield the lower bound values. There are several options

for creating the upper bound lookup table. First, if the "preprocessor" *g(x)* is used, the same lookup table can be used for both the low and high estimates. However, this is only guaranteed to give reliable bounds for monotonic functions, as highlighted in Figure 8.2. Second, each *m*-bit truncated input can be incremented by one and used to calculate a high-precision solution, which can then be rounded up. Unfortunately, this method can also fail near stationary points. Third, prior knowledge about the maximum gradient of the function can be used to properly offset the two lookup tables. For example, since the sine function has a maximum slope of +1, the low and high estimates should differ by only a single bit in their least significant digit. Finally, in regions of negative slope, the upper and lower bound lookup table values should be swapped.

Even with lookup tables, great care must be taken near stationary points. If the rounded up and rounded down versions of the precise input value straddle a stationary point, the function should be evaluated at both points. If the function's second derivative is negative (i.e., local maximum), the lesser of these two solutions should be assigned as the low estimate. In this case the high estimate can be computed by adding the function's maximum slope value to the low estimate. On the other hand, if the function's second derivative is positive (i.e., local minimum), the greater of the two solutions becomes the high estimate and the low estimate is computed by subtracting the maximum slope value.

Continuing with the sine function example, Figure 8.4 shows three sample points along the curve. The left and right frames show the local min/max at $-\pi/2$ and $+\pi/2$, and the middle frame shows the function at $-\pi/8$. In this figure, the points on the horizontal axis are determined with 32-bit and 8-bit precision for the precise and low/high inputs, respectively. Note that the low/high inputs in the left and right frames straddle the stationary points. Again, yellow block arrows indicate the desired proper error bounds. In this case, the proper error bounds always span the same vertical distance since the upper/lower bounds should differ by just one digit in the LSB position of the 8-bit estimates (recall that the sine function's maximum slope is 1).

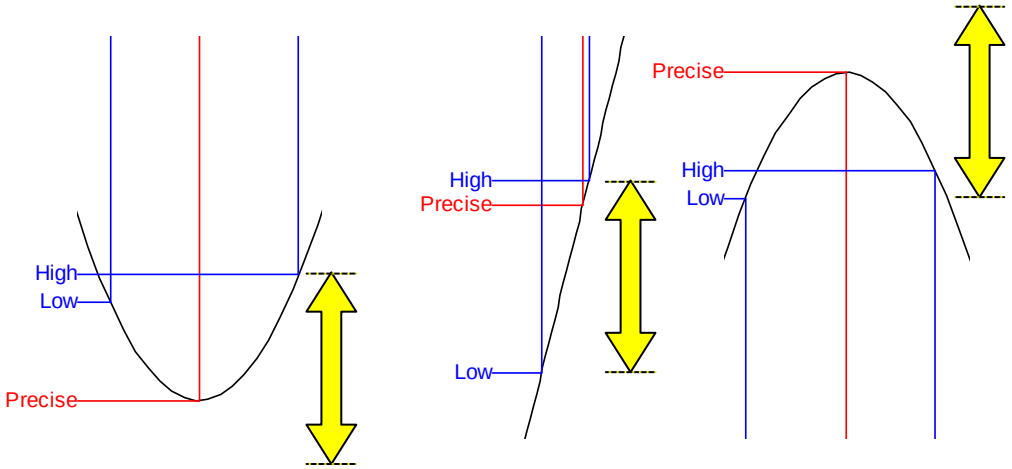


Figure 8.4 Approximating Sine Function at Three Sample Points

Table 8.1 presents an initial attempt at generating entries for the low/high estimate lookup tables. The decimal values correspond to the data points shown in Figure 8.4 above, whereas the hexadecimal numbers show the binary equivalent of each value rounded to the appropriate precision (following the two's complement number system defined in Table 8.2 below). As the figure demonstrates, there is some confusion in the left and right frames since the precise solution falls outside the bounds set by the rounded input values. Likewise, the table shows trouble near $\pm\pi/2$ since the 8-bit truncated output values allow no tolerance for the precise solution.

x		Input		Output	
-π/2	*Precise*	-1.570796327	0x9B7812AF	-1.000000000	0xC0000000
	Low	-1.578125	0x9B	-0.999973	0xC0
	High	-1.562500	0x9C	-0.999966	0xC0
-π/8	*Precise*	-0.392699081	0xE6DE04AC	-0.382683432	0xE7821B5A
	Low	-0.406250	0xE6	-0.395167	0xE6
	High	-0.390625	0xE7	-0.380766	0xE7
+π/2	*Precise*	1.570796327	0x6487ED51	1.000000000	0x40000000
	Low	1.562500	0x64	0.999966	0x3F
	High	1.578125	0x65	0.999973	0x3F

Table 8.1 Initial Attempt at Generating Lookup Tables for Sine Function Approximation

Precise	Bit#	31	30	29	28	27	26	25	24	…	1	0
	Value	-2^{1}	2^{0}	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	…	2^{-29}	2^{-30}

Estimates	Bit#	7	6	5	4	3	2	1	0			
	Value	-2^{1}	2^{0}	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}			

Table 8.2 Fixed-Point Two's Complement Number Formats for Example in Figure 8.4

The steps outlined earlier in this section aid in resolving these problems. The second derivative is positive near $-\pi/2$, so the procedure says to take the larger value as the high estimate (0xC0) and subtract one to get the low estimate (0xBF). There is no problem near $-\pi/8$, so those lookup table entries are unchanged. Near $\pi/2$ the second derivate is negative, therefore the smaller value is assigned to the low estimate (0x3F) and the high estimate is incremented from that value (0x40). Note that in all three cases the precise solution is within the error bounds defined by these low and high estimates. In fact, because the sine function has a maximum slope of 1, the 8 MSBs of the precise solution will match either the low or high estimate for every input value. Also note that the block *g(x)* is not needed because the low and high estimate lookup tables can be indexed using only the 8 MSBs of the precise input value.

x		**Input**	**Output**
-π/2	*Precise*	0x9B7812AF	0xC0000000
	Low	0x9B	0xBF
	High	0x9C	0xC0
-π/8	*Precise*	0xE6DE04AC	0xE7821B5A
	Low	0xE6	0xE6
	High	0xE7	0xE7
+π/2	*Precise*	0x6487ED51	0x40000000
	Low	0x64	0x3F
	High	0x65	0x40

Table 8.3 Improved Lookup Table Entries for Sine Function Approximation

3. Improved Method for Generating Lookup Table Estimate Values

The preceding sections highlighted some of the challenges of developing approximate solutions to numerical functions and showed how lookup tables can

overcome some of these challenges. This process can be made even more robust using the procedure described below to generate the lookup table contents. Through this method, lookup table approximations can be applied to a broader class of functions, thereby eliminating some of the restrictions imposed earlier in this chapter.

The basic concept for this alternative approach is that the off-line calculations generating the lookup table contents can incorporate maximum/minimum detection over the entire input range between "low" and "high." Instead of calculating the function at only the "low" and "high" rounded input points, the function is evaluated at every point in between that could be input to the precise module in Figure 8.3. Referring to Figure 8.3, there are 2^n of these intermediate input values for each m-bit truncated input value. As these points are evaluated starting at the low input value and moving towards the high input, max/min variables are stored and updated as appropriate. Once the high input value is reached, the minimum output is rounded-down (truncated) and the maximum output is rounded-up. These values are stored in the low and high lookup tables, respectively. No matter where in this range the precise input value actually exists, the upper/lower estimates will properly bound the precise output.

This concept is shown in Figure 8.5 below, where the low and high inputs differ by one bit in their LSB position and the intervening points are spaced according to the bit-length of the precise input vector. For this hypothetical function, neither the low nor high input values yield useful error bounds. The horizontal red lines mark the maximum and minimum values of the function over the range low to high, thus these extrema should be entered into the low and high lookup tables.

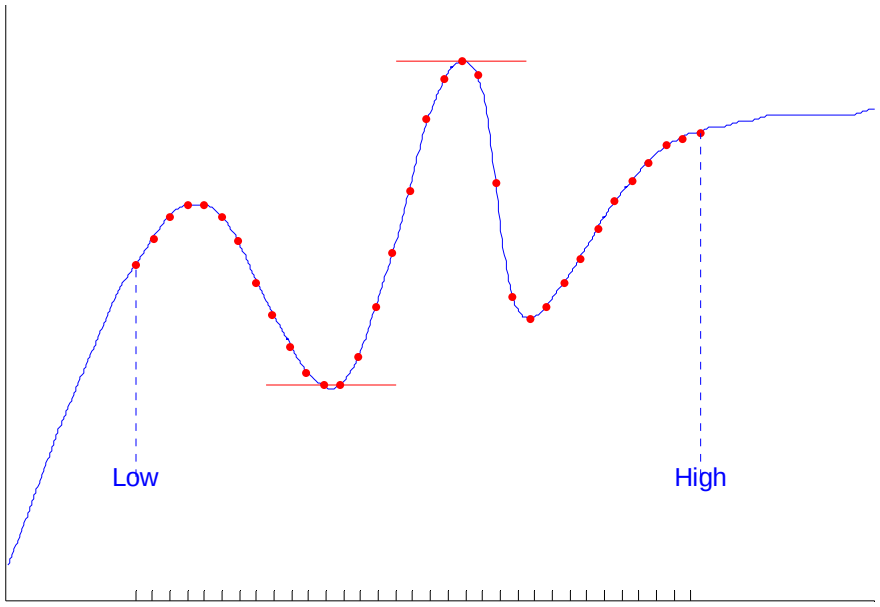


Figure 8.5 Generating Lookup Table Error Bounds for Arbitrary Functions

This approach requires no assumption about the type of function being evaluated and can easily handle discontinuities and stationary points. One simply computes each of the 2^n unique input values for each m-bit MSB sequence and records the max/min values. These max/min values are then rounded up and down before being entered into the high and low estimate lookup tables. However, the computation time required to calculate and compare 2^n inputs for all 2^m lookup table entries can be burdensome. For example, using MATLAB (running on a 1.86 GHz Pentium machine) to compute every sine function result when $m+n = 24$ bits, takes roughly 12 seconds. It would take over 50 minutes to do these calculations when $m+n = 32$ bits. Nonetheless, the upper/lower lookup tables only need to be generated once for a particular combination of m and n precision levels. Thus, when practical, the procedure described above is perhaps the most effective way of creating the RPR error bounds lookup tables.

4. Designing a Voter to Compare Precise and Approximate Calculations

Although the notion of checking whether a high-precision solution is within particular error bounds seems intuitive, implementing this error check in computer hardware requires careful design. In particular, for FPGA designs it is possible for any component in the circuit to suffer SEUs. An important aspect of RPR designs is that the

physically-larger precise-computation module is more likely to be affected by SEUs than the smaller redundant modules. Nonetheless, the error bounds are susceptible to faults. Figure 8.6 shows various fault conditions in an RPR circuit, assuming that only one module can be faulty at any given time.

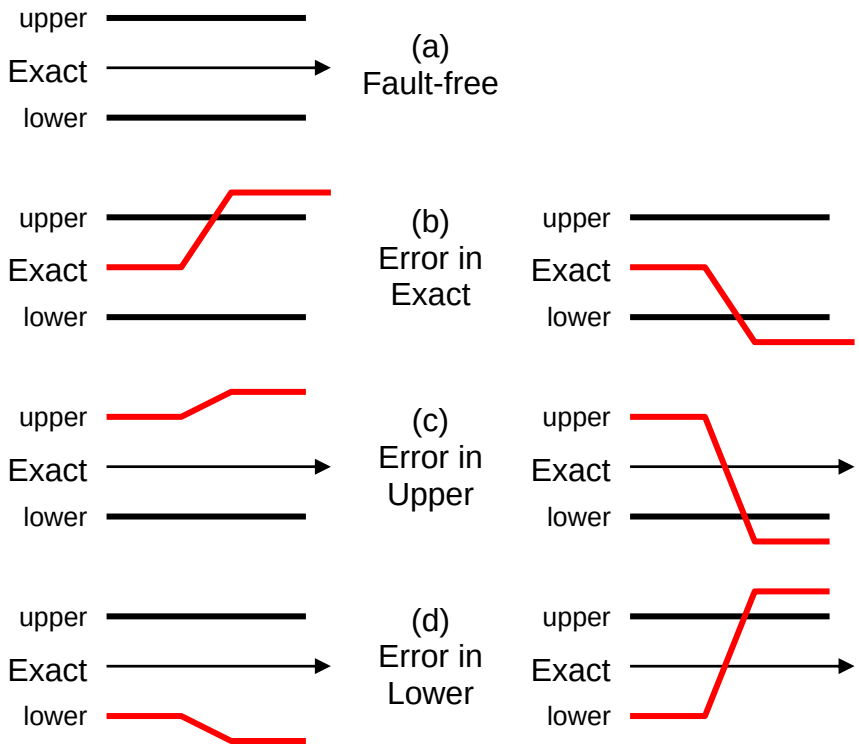


Figure 8.6 Possible Error Conditions in RPR

In the fault-free situation (a), the voter finds the exact solution is within the error bounds and forwards this result to the output. When the exact solution is faulty and falls outside the bounds (b), a logical option is for the voter to output the midpoint between the lower and upper bounds. Some faults to the upper/lower bounds cause data errors, while others may go undetected. The left diagrams for situations (c) and (d) show cases where the fault is transparent to the voter, since the exact solution falls within the bounds. However, the right diagrams for (c) and (d) show situations the voter will detect. However, when the upper and lower bounds are incorrect relative to each other, the voter assumes a single fault has upset the error bounds and outputs the exact solution. Figure 8.7 presents pseudo-code describing the voter behavior in an RPR design. Again, it is

important to point out that this voting methodology assumes there is never more than one faulty module in the circuit.

```
if ( exact(MSB) >= lower ) and ( exact(MSB) <= upper )
        output = exact
else if ( upper <= lower )
        output = exact
else
        output = lower(MSB) + 1/2 (upper-lower)
end
```

Figure 8.7 Pseudo-code for RPR Voter

This voting procedure is simplified further if the function has a maximum slope of one. In this case, the upper and lower bounds always differ by the equivalent of one bit in the least significant digit. This is because the inputs to the low/high modules differ numerically by the value of one in their LSB position, as mentioned in the previous section. For the exact solution to fall within the error bounds, its most significant bits must correspond to the bits in either the upper or lower bound. Thus the voter only needs to check whether the MSBs of the exact solution equal either the lower or upper bound.

C. CONSEQUENCES OF IMPRECISION

Though system engineers generally desire maximum possible precision in mathematical computations, compromises must be made between the costs and benefits of high precision. In many applications, lower precision calculations act like a source of noise, causing a degradation in quality similar to sensor noise, environmental disturbances, and other noise sources. Most systems can tolerate a limited amount and/or duration of noise. Those that cannot are generally not robust enough to use in real-world applications. This section demonstrates that occasional lower precision results caused by radiation-induced SEUs in an RPR system are nearly undetectable in certain applications. Image processing serves as an example of a useful function in which RPR and the CORDIC algorithm can be used effectively. In this application, RPR is superior to alternatives such as TMR or an SEU-intolerant design.

1. Scenario Description

The scenario investigated here is an extension of the satellite image processing example presented in Chapter IV. The engineer's task is to build an image compression circuit in order to minimize downlink bandwidth consumption. FPGAs are well-suited for this task for several reasons. In addition to being significantly less expensive than custom ASIC chips, they are readily available and can be quickly integrated into the spacecraft system. Furthermore, the ability to load different circuit configurations in real-time will enable dynamic optimization of the image compression algorithm.

The image compression processor is tasked with reducing the number of bits-per-pixel necessary to reconstruct single frame images collected by an onboard camera. The camera is a 512x512 pixel panchromatic imager operating at a nominal 1 Hz frame rate. Each pixel's gray scale intensity level is represented with 16-bits of dynamic range. In order to support continuous image collection and transmission, the image compressor must process over 262,000 pixels per second (or 3.81 micro-sec per pixel). These data rates are easily supported by current FPGA technology, which can operate at clock speeds over 100 MHz.

However, there is concern about the SEU tolerance of FPGAs since their configuration logic is susceptible to radiation-induced upset. Therefore, the engineer must develop a fault-tolerant design while minimizing the area and power consumption. There are three main goals for fault tolerance in this situation. First is the detection of errors. Without some mechanism for detecting errors, SEUs can cause the satellite to transmit corrupted imagery indefinitely. By including error detection on the spacecraft, the system can either automatically take corrective action or notify operators on the ground so they can initiate recovery efforts. Second, in conjunction with error detection it is desirable for the system to perform some error correction until corrective measures can successfully remove any SEUs. This ensures that the satellite continues to provide images of satisfactory quality even when SEUs affect the circuit. Finally, there must be some means of fault correction, which for FPGAs involves rewriting the configuration memory contents. Error detection accelerates the fault correction step by identifying the region(s) affected by SEUs and focusing recovery efforts toward those components. All three of these goals are supported by the RPR approach.

2. Image Compression with Discrete Cosine Transform

One of the most common data compression techniques involves the discrete cosine transform (DCT). It has been widely used in image and video compression applications [104] for reducing bandwidth consumption in transmission and minimizing data storage requirements. DCT is one of the methods used in the well-known JPEG and MPEG standards [25], [63]. Many satellites, both government and commercial, perform image collection and dissemination. Thus there is significant practical interest in examining the reliability and performance of a DCT algorithm for space applications.

The DCT has been successfully implemented in VLSI, PLD and FPGA hardware technologies [104], [105], [106]. Several architectures have been proposed in which CORDIC processing elements are used to calculate the DCT [92], [93], [107]. By appropriately setting the *x*, *y*, and *z* inputs, the CORDIC algorithm can perform multiplication in conjunction with calculating sine and cosine functions. Using this approach, CORDIC is a particularly efficient method of computing the DCT.

Finally, it is important to recognize that digital systems are incapable of representing real numbers with absolute precision [63]. Thus the DCT, which operates upon real numbers, is well-suited to various approximation techniques, and RPR is an appropriate method of achieving fault tolerance for DCT designs.

a. Background

The DCT, related to the discrete Fourier transform, was first developed in 1974 by Ahmed, Natarajan, and Rao [108]. It's original formulation applies to one-dimensional signals, but can be extended to the two-dimensional case. The basic formulas describing the two-dimensional forward (G_{uv}) and inverse (g_{mn}) DCT, assuming an original signal matrix of size *MxN*, are given in Equation 8.1 [104].

$$G_{uv} = \frac{2c(u)c(v)}{\sqrt{MN}} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} g_{mn} \cos\left[\frac{(2m+1)u\pi}{2M}\right] \cos\left[\frac{(2n+1)v\pi}{2N}\right]$$

$$g_{mn} = \frac{2}{\sqrt{MN}} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} c(u)c(v)G_{uv} \cos\left[\frac{(2m+1)u\pi}{2M}\right] \cos\left[\frac{(2n+1)v\pi}{2N}\right] \quad (8.1)$$

$$\text{where } c(k) = \begin{cases} \frac{1}{\sqrt{2}} & \text{if } k = 0 \\ 1 & \text{otherwise} \end{cases}$$

In principal, the DCT can be computed for an entire image in a single pass. However, all practical implementations involve processing the image as a series of square subarrays, most commonly blocks of 8x8 pixels. Studies have found that larger subarrays generally do not show better data compression performance, as the correlation among neighboring pixels is typically limited to small regions of the image [104]. Since the variables *m*, *n*, *u*, and *v* in Equation 8.1 can only take on integer values, the cosine terms can be precomputed and stored in lookup tables. Smaller block sizes allow compact lookup tables for storing these constants and smaller memories for storing subarray elements during the computation.

Figure 8.8 shows the 64 basis functions that comprise the DCT calculation for the 8x8 case. Each sub-image corresponds to a single element of the transformed domain block. If the transformed domain coefficients are specified with sufficient accuracy, any 8x8 block of pixels in the image domain can be reconstructed from some combination of these basis functions. It is important to note that the DCT calculation does not by itself provide data compression. Rather, compression is achieved through quantization and/or removal of components from the transformed domain. Numerous quantization and data coding techniques, often involving adaptive algorithms, have evolved for compressing the transformed image representation with minimal distortion of the restored images [104]. For example, in many cases the coefficients corresponding to high spatial frequencies can be discarded with virtually no perceptible degradation in the image.

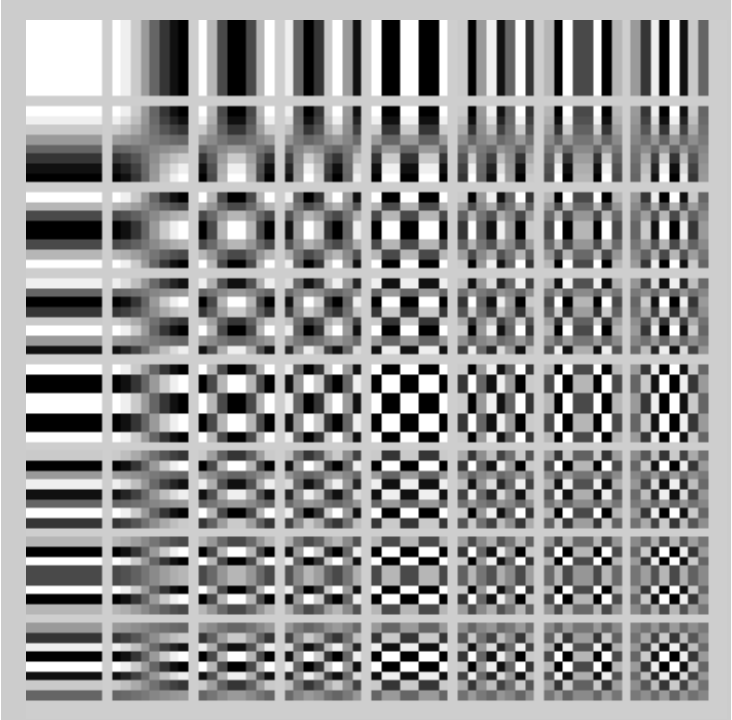


Figure 8.8 Basis Functions for 8x8 DCT

b. Measuring Image Quality

Assessing image quality is inherently subjective. Rao notes that the human viewer "is the ultimate judge regarding the quality of the processed images." [104] Nonetheless, various quantitative measures are commonly used to assess image quality. These metrics allow objective comparison of different image processing methods. Furthermore, in applications such as medical diagnostics, images that "look good" to human observers may actually be inferior if they are processed in such a way that critical information is lost.

Two of the most common image quality metrics used in the literature are mean squared error (MSE) and peak signal-to-noise ratio (PSNR). These metrics are defined in Equation 8.2. They both quantify how "noisy" a reconstructed (and possibly corrupted) image is compared to the original image.

$$MSE = \frac{1}{MN}\sum_{m=0}^{M-1}\sum_{n=0}^{N-1}\left[g(m,n) - \hat{g}(m,n)\right]^2$$

$$PSNR = 10\log_{10}\left(\frac{\left(2^B - 1\right)^2}{MSE}\right) \quad \text{where } B = \text{bits per pixel} \tag{8.2}$$

c. MATLAB Testbed

A MATLAB simulation testbed was developed to examine the quality of DCT-based image compression under various levels of numerical precision and error rates (see Appendix B). The "signal processing" toolbox within MATLAB includes the DCT and inverse-DCT (IDCT) functions. For the purposes of this testbed, however, it was necessary to implement the DCT operations directly rather than use the built-in functions. This permitted manipulation of numerical precision over small regions of individual images and at specific places in the DCT computation sequence.

For a satellite imaging application, only the forward DCT processing is performed on-board and therefore susceptible to errors. Image reconstruction with the IDCT is performed on the ground and assumed to be error-free. The DCT algorithm was modified to allow manipulation of the numerical precision and accuracy of the DCT coefficients. Complete processor failure, as might occur in unprotected TMR or RPR voter circuitry, is modeled with a random number generator. In this situation the coefficients produced by the circuit can take on any value between +/- N*(2^B-1), where B is the number of bits per pixel and (2^B-1) is the largest possible pixel intensity. Imprecision is modeled by discretizing and rounding properly calculated DCT coefficients.

In addition, fault persistence was modeled by considering the time required for partial and/or full FPGA reconfiguration. Data from the CFTP flight experiment hardware suggest full device reconfigurations take approximately 70 msec and single-frame partial reconfigurations require roughly 24 μsec. These time durations were converted into the number of sequential pixels flowing through the processor until potential faults can be corrected. For this scenario with a 512x512 pixel imager operating continuously at 1 Hz, 8 pixels are processed in the time it takes to refresh a single configuration frame on the FPGA and 18,000 pixels are processed during a full device reconfiguration. In either case, if on-chip error detection is available, only a small portion of a single frame will be corrupted before the system is restored. For larger images and/or faster frame rates, the throughput would necessarily be higher and more pixels would be corrupted before the device could recover from error-producing SEUs.

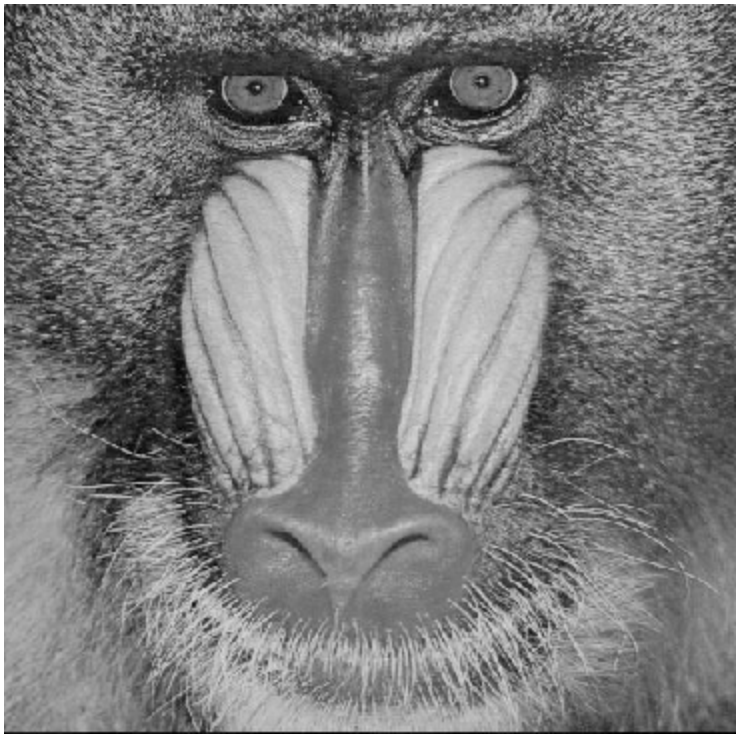
These fault models provide information regarding the potential consequences of SEUs in FPGA-based DCT processing. However, they are not based on any particular design implementation. More sophisticated fault and algorithm models can be easily integrated into this preliminary testbed to investigate the performance of specific designs. When building circuits for real-world implementation, one could use the actual design architecture(s) to more precisely model the SEU response of the image processing system. In addition, once the design matures enough for testing on real circuits, the SEU simulator from Chapter VI could be used in conjunction with the MATLAB testbed to improve these predictions.

3. Effect of Imprecision on Image Quality

This scenario was designed to determine whether imprecise computations from an SEU-affected RPR design cause noticeable and/or unacceptable results. In addition, it illustrates the impact of SEUs that defeat a circuit's fault tolerant capabilities (TMR, RPR, or otherwise). A TMR design may have less frequent failures than an RPR version, but TMR failures have greater likelihood of causing gross miscalculations. Furthermore, the TMR failures may persist for a longer duration since the larger circuit takes more time to reconfigure. During the fault repair time, the processor will continue to provide erroneous results, degrading a larger region of the image.

The first example examines what happens when circuit failure causes the calculation to produce random DCT coefficients. Figure 8.9 shows the before and after images for such a fault persisting for 70 msec and affecting the calculation of over 18,000 coefficients, or about 7% of the total pixels. As expected, the image recreated upon inverse transforming these random coefficients is completely corrupted in the regions that were processed while the fault persisted. For a system performing continuous configuration readbacks and/or scrubbing on the FPGA device, this example can be considered a worst case since 70 msec is the longest duration that an SEU would persist. For systems with less frequent readback/scrubbing or systems that don't include on-board fault detection/correction, it is possible for SEU-induced faults to spoil entire images or sequences of images.

Original Image



Restored Image

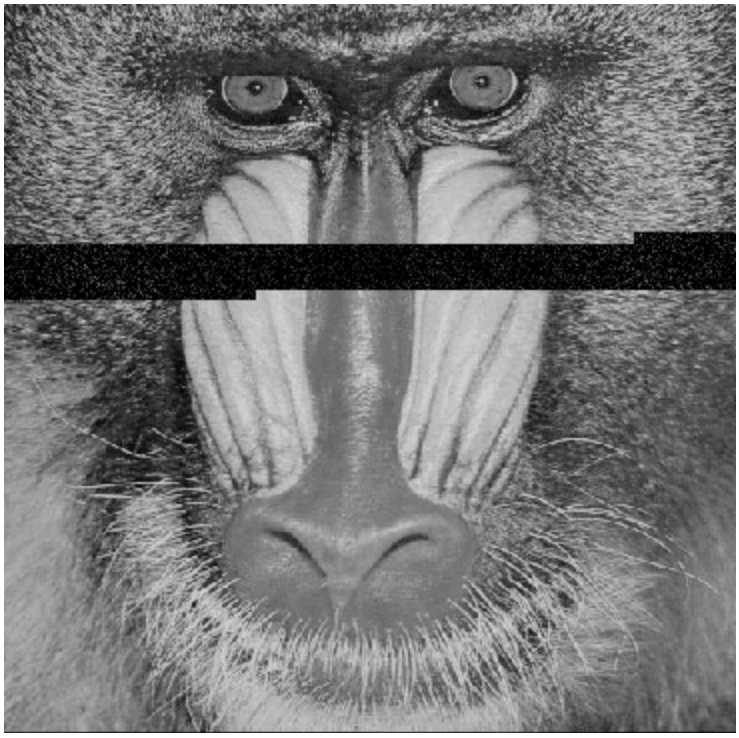


Figure 8.9 “Baboon” Image with Temporary Failure of DCT Processor

The next example looks at the effect from imprecise coefficient calculations over the same 70 msec fault duration. Again, roughly 7% of the pixels are affected. In this case each DCT coefficient in the error zone is approximated by an 8-bit number, compared to the 32-bit representation of coefficients outside the error zone. A sample run comparing the fault-free and faulty images is shown in Figure 8.11. The corruption is virtually undetectable to the casual observer. Figure 8.10 shows the difference between the two images, identifying a horizontal band of distortion in the upper portion of the image. Careful examination of Figure 8.11 reveals some blurring, especially in the central region of trees. This region is expanded in Figure 8.12. Nonetheless, for most applications such minimal degradation is acceptable.

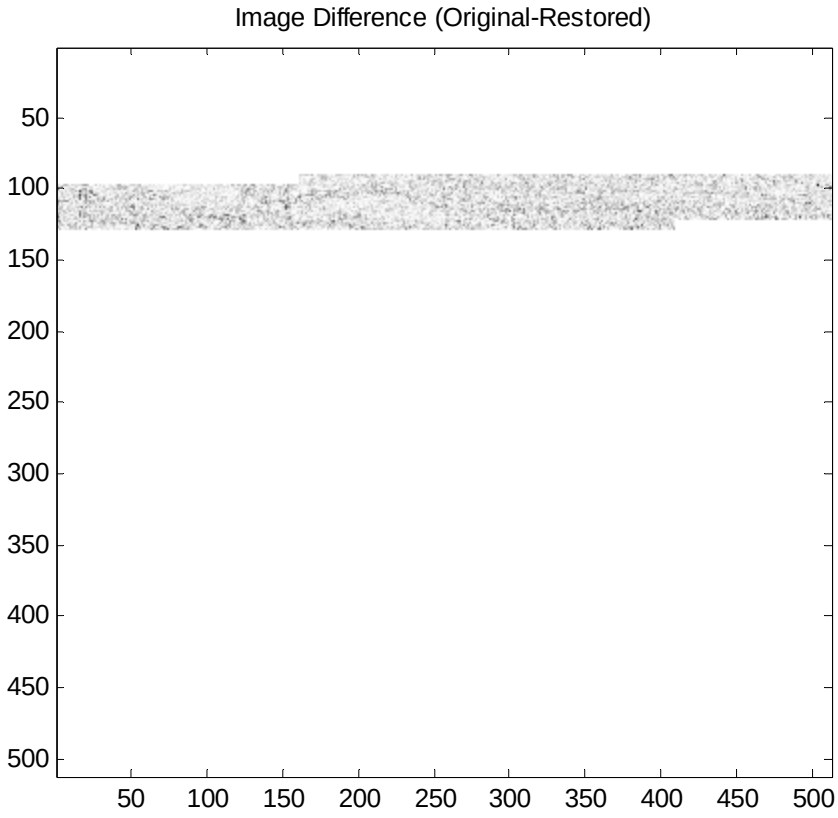


Figure 8.10 "Gold Hill" Difference Image

Original Image

Restored Image

Figure 8.11 “Gold Hill” Image with Temporary Imprecision in DCT Processor

Original Image

Restored Image

Figure 8.12 Detail of “Gold Hill” Image with Temporary Imprecision in DCT Processor

To further demonstrate the inherent robustness of this image processing scenario, the final example shows what would happen if an entire image was processed with reduced precision. This would be useful information when determining the appropriate level of detail for the full precision modules in TMR and RPR structures. Furthermore, various factors may cause SEU-induced imprecision in an RPR design to persist over more than the ~18,000 pixels assumed in the previous examples. For example, if the camera system were to operate at standard video rates of 30 frames per second instead of the 1 Hz rate assumed in this scenario, 70 msec of error persistence would translate into 550,000 pixels, or more than two entire images. As Figure 8.13 shows, at this level of imprecision the image is beginning to appear somewhat "blocky," although overall the degradation is slight. Figure 8.14 shows a more detailed view, demonstrating the blurring that occurs when 32-bit coefficients are approximated by 8-bit values.

Original Image

Restored Image

Figure 8.13 “Lena” Image with Persistent Imprecision in DCT Processor

Original Image

Restored Image

Figure 8.14 Detail of “Lena” Image with Persistent Imprecision in DCT Processor

In addition to subjectively judging the preceding images, one can use the image quality metrics from Equation 8.2. Table 8.4 lists the mean squared error and peak SNR metrics for the three examples described above. The left column of data, labeled "Fault-Free," is created by performing a 32-bit DCT calculation, followed immediately by the IDCT operation. The right column of data corresponds to the hypothetical fault conditions described for each example. Although these mathematical metrics indicate considerable distortion between the fault-free and faulty images, only the "Baboon" example appears obviously corrupted to the average viewer. Distortion in the two examples of imprecision are practically indistinguishable to human observers viewing these images through typical media (hardcopy and computer display).

	Fault-Free		Faulty	
	MSE	**PSNR**	**MSE**	**PSNR**
"Baboon"	$3.52*10^{-13}$	172.7 dB	1,581	16.1 dB
"Gold Hill"	$3.54*10^{-13}$	172.6 dB	2.07	45.0 dB
"Lena"	$3.54*10^{-13}$	172.6 dB	26.7	33.9 dB

Table 8.4 Image Metrics for Figure 8.9-Figure 8.13

As this section demonstrates, RPR is a viable option for many image processing applications. The simulation process developed above serves as a practical example of how one can investigate the performance of an RPR architecture in an SEU environment. By modeling a system and the potential faults that might affect it, one can better understand the consequences of particular fault-tolerant approaches. The examples above were based on simple models of a DCT processor. As a design matures, more detailed models can be developed to improve the accuracy of such studies. Using this process even in the preliminary design stages allows one to make informed decisions about how best to achieve fault tolerance.

4. Affect on Total Performance Metric (TPM)

The previous section demonstrated that image processing tasks are often fairly robust in handling numerical imprecision. It was shown both qualitatively and

quantitatively that the risk of imprecision due to an RPR architecture is minimal. While Figure 8.9 showed that incorrect data can dramatically corrupt images, Figure 8.11-Figure 8.14 showed that even significant loss of numerical precision (e.g., from 32-bit to 8-bit precision) causes only slight distortion. This information is vital to determining a system's total performance metric (TPM).

As described in Chapter IV, the TPM process is useful for guiding system development based on quantitative measures of cost and benefit factors, and scaling factors to relate these diverse metrics. For the problems addressed in this dissertation, the TPM factors include size, speed, power, reliability, and precision. The goal of TPM is to determine the optimal design solution that maximizes the function:

$$TPM = Benefit - Cost = \sum_{i}^{N_{benefits}} K_{B_i} B_i - \sum_{j}^{N_{costs}} K_{C_j} C_j \tag{8.3}$$

The scaling factors K that appear in the equation above represent the relative importance of each cost and benefit term. Based on the results from the previous section, it is clear that $K_{reliablitity}$ should be larger than $K_{precision}$, since failure of the image compression algorithm has much more severe consequences than reduced-precision data. This conclusion is, of course, scenario-dependent. For applications such as video broadcasts of entertainment content or satellite imaging for broad-area land surveys, it is safe to assume that the precision factor is much less important than reliability. On the other hand, applications such as astronomical imaging with the Hubble Space Telescope may be much less tolerant of images corrupted by imprecise data. The following discussion addresses situations in which the precision is less important than reliability.

Example 2 from Chapter IV considered the TPM for a notional satellite image processor. The assumptions made about the reliability and precision terms in Chapter IV can be improved by using simulation tools such as those used in the previous section for demonstrating the effects of data errors and imprecision. Figure 8.15 shows the behavior of these two TPM factors based on results from the previous section. Referring to the right frame in the figure, note that there is minimal loss of "benefit" as precision is reduced from the design specification value of 16 bits to 10 bits. Similarly, there is no benefit to increasing the precision above 16 bits. There is, however, a critical point where the precision factor drops off quickly, since image reconstruction using extremely

coarse data is nearly the same as using incorrect data. Experimentation with the images used earlier in this chapter, showed that this transition region occurs between 5 and 10 bits. At 5 bits of precision or lower, the images were almost indecipherable and therefore provide essentially no benefit. On the other hand, at 10 bits of precision the image degradation was barely perceptible and certainly not significant for entertainment and similar purposes. The circle on the graph marks the point where 11 bits of precision provides 95% of the maximum precision benefit value. Sample images processed at this precision level yielded a peak SNR (PSNR) value of approximately 45 dB.

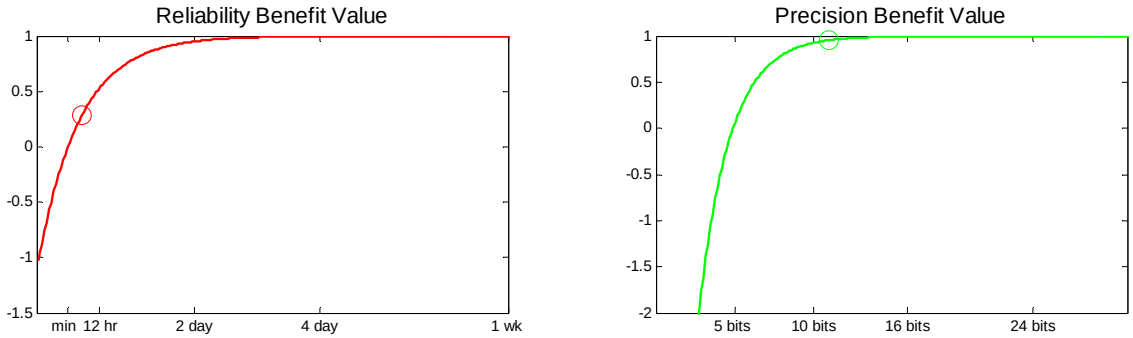


Figure 8.15 Benefit Value Functions for Reliability and Precision Factors

The reliability term shown in the left frame exhibits a similar roll-off at lower reliability values and an plateau region for higher values. The x-axis is measured by mean time between error (MTBE), where an error corrupts one entire image frame. Based on this definition of reliability, it is assumed that one or more bad images every minute is unacceptable, thus the benefit value for a one minute MTBE is zero. Considering a video entertainment application, one error every hour is presumably noticeable, but not catastrophic. Therefore, this MTBE provides about half the desired reliability value. At the upper extreme, an MTBE of two days or longer provides nearly the maximum reliability benefit. MTBE can be converted into an average pixel error rate, and vice versa. The example in Chapter IV describes a system with a 512x512 pixel camera that operates at 1 frame per second, or 86,400 frames per day. To achieve the same PSNR level of 45 dB as discussed earlier, each 512x512 image must have only 14

pixel errors. This translates into 4.6 bad images per day, or an MTBE of 5.2 hours. This point is marked on the graph by a red circle.

By combining the data in these two graphs, one can determine the proper ratio between $K_{reliablitity}$ and $K_{precision}$. For this example, this can be accomplished by scaling the two functions such that the PSNR=45 dB points coincide. Figure 8.16 shows this combined plot. The intersection of these two curves occurs at Precision=11 bits and MTBE=5.2 hours. From these curves it is determined that the asymptotic benefit value for reliability (MTBE) is 3.3 times that of precision. Thus the K factors should be set such that $K_{reliablitity} = 3.3 \times K_{precision}$.

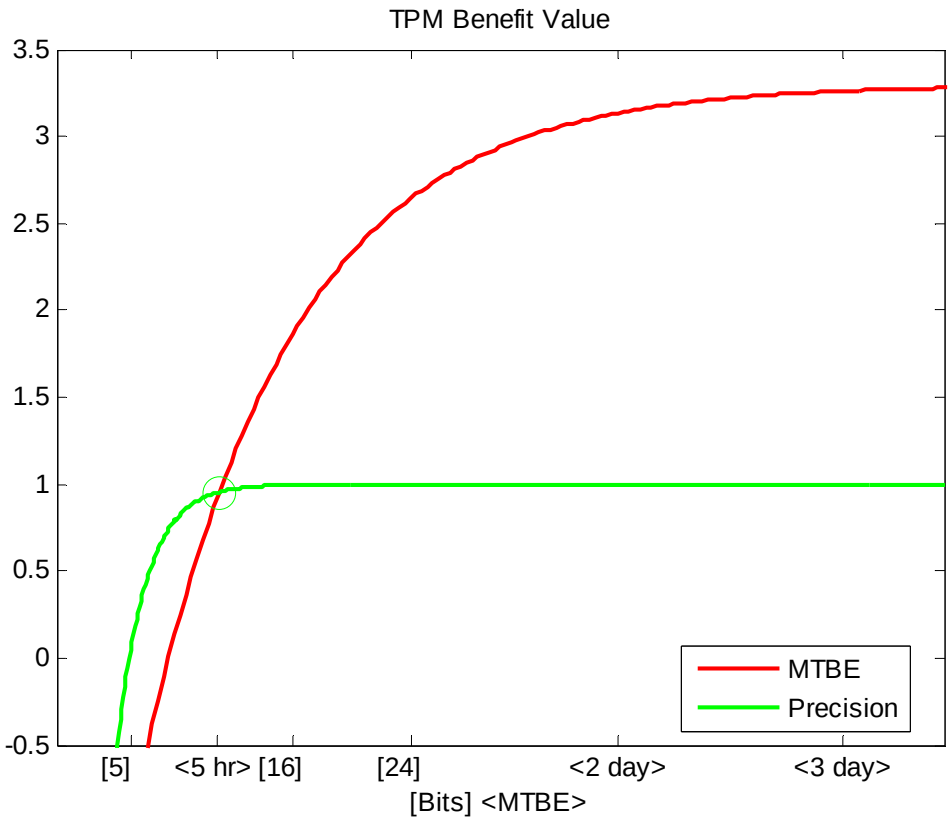


Figure 8.16 Relationship Between Precision and Reliability TPM Factors

Example 2 from Chapter IV also considered power, area, throughput and latency. Throughput was determined to be the single most important factor in that example, whereas power, area and latency were much less significant. Therefore the next step in

the TPM process is to develop an analysis technique for relating throughput to reliability and/or precision. One possible approach is to use surveys and controlled experiments of human viewer reactions to varying video rates (throughput) as compared to different levels of picture clarity (precision) and pixel or frame errors (reliability). This would yield the TPM curve for throughput and would reveal the relative importance (*K* factors) among these three metrics. Using the TPM curves and scaling factors for these three most important parameters allows one to determine an optimal solution that best combines speed, accuracy and reliability. Similar analysis can be carried out for the other TPM factors applicable for a given application. As size, power, latency or other parameters increase in importance, they can also be integrated into the TPM.

This example highlights that both subjective and objective criteria are often needed to relate the various TPM factors. For example, while mathematical image quality metrics such as PSNR are useful, determining the "annoyance threshold" for bad images during video entertainment broadcasts is very subjective. Nonetheless, especially in image processing applications, these subjective measures are often more important from a system-level perspective. TPM provides a simple way to integrate both qualitative and quantitative measurements of each design parameter. Armed with the information generated by the TPM process, a system engineer can decide which design solution best achieves all of the design goals.

THIS PAGE INTENTIONALLY LEFT BLANK

IX. CONCLUSION

A. SUMMARY OF RESEARCH

This research investigated methods of building and testing flexible, reliable and efficient spacecraft computer systems. Recent advances in FPGA technology make these devices well-poised to become key components in future satellites that will likely require flexible high-performance computing systems. Spacecraft computers must not only meet strict size, weight and power constraints, but also operate reliably in harsh radiation environments. The major reliability concern addressed in this research is the effect of radiation-induced SEUs on FPGA circuits. While traditional approaches to fault tolerance assume reliability is the preeminent concern, this research highlights the multitude of issues that a spacecraft engineer must address. Specifically, power consumption is identified as a critical resource that must be carefully balanced against reliability goals when designing spacecraft computers.

A major focus of this research was developing the RPR architecture as an efficient means of achieving fault tolerance for FPGAs. RPR is an alternative to the more common TMR method that has long been used to protect systems against SEUs. RPR's primary advantages include reduced area requirements and power usage. Certain computational problems, such as the primitive logic operations of AND, XOR, etc., cannot be formulated in terms of approximate calculations and are not amenable to RPR. These were labeled Class B problems, in contrast to Class A problems, which can be used in an RPR architecture. Since most numerical functions and physical systems have some tolerance for imprecision and/or noise, RPR has broad applicability in real-world systems.

The concept of a total performance metric (TPM) was introduced to evaluate the relative merit of RPR and other approaches such as TMR. TPM takes into account the importance of various design parameters and aids the system engineer in determining the optimal solution to a given problem. The methodology for developing this TPM can easily be tailored for use in various design scenarios. The main concept in the TPM approach is that each design parameter can be expressed as either a cost or benefit

function and related to other parameters by means of simple scaling factors. Several notional examples were developed in which power, fault tolerance, and other factors were integrated via TPM to identify the best solution. Although the behavior and relationships among the TPM terms vary between applications, this methodology is a powerful tool for improving the system design process. TPM factors related to image quality were examined in detail for a specific image processing algorithm to demonstrate the real-world applications of TPM.

Several CORDIC designs for calculating sine and cosine functions were used as test cases for detailed investigations of the SEU tolerance and power consumption in actual FPGA devices. CORDIC was chosen as a good example of a moderately complex function fitting the description of a Class A algorithm appropriate for RPR. The circuits were implemented on the Xilinx Virtex XQVR600 FPGAs contained in the CFTP experiment. In addition, some CORDIC designs were used in radiation testing of a prototype Virtex-II board that included an XC2V6000 device.

TMR and RPR versions of these CORDIC circuits were evaluated through SEU simulations at NPS and proton radiation testing conducted at UC Davis' Crocker Nuclear Laboratory. Although radiation testing did not provide comprehensive characterization of these two fault tolerant approaches, it yielded valuable data for validating the custom-built CFTP SEU simulator. In addition, radiation testing demonstrated that the SEU detection and correction techniques of the CFTP experiment were functional and ready for space flight. Hundreds of SEUs were detected by the automatic FPGA configuration checking circuitry on the CFTP board. The SEUs identified during several hours of radiation testing are equivalent to the number of SEUs expected in roughly one year of normal radiation conditions for the CFTP space experiment.

The SEU simulator provided a means of comprehensively evaluating individual FPGA designs by exhaustively testing the SEU susceptibility of every configuration bit on the chip. Extensive testing was conducted with TMR, RPR and unprotected CORDIC circuits. The data demonstrated that the RPR approach provides fault tolerance comparable to that of TMR. The RPR and TMR designs had residual SEU sensitivities of 0.45-1.94% and 0.34-0.89% probability of SEU-induced output data error,

respectively. Overall, TMR was roughly twice as effective as RPR in preventing SEU-induced data errors, which was expected due to TMR's more comprehensive redundancy structure. RPR was most effective in protecting the most significant bits in a circuit's output, which for these test circuits were the 8 MSBs. Errors in the 8 MSBs of the RPR circuits were about as likely as errors anywhere in the output data word of the TMR circuits. Thus, RPR is an effective method of protecting against configuration bit SEUs.

Power simulations were conducted to quantify the power savings of RPR compared to the TMR architecture. Using commercial software tools (ModelSim and XPower) and established procedures, static and dynamic power consumption were determined for the same circuits used for SEU testing. RPR required less than 5% more power than the unprotected circuits. The TMR circuits used about twice as much power as the RPR circuits. Therefore, RPR offers dramatic improvement in power usage over the traditional TMR approach. Coupled with the SEU simulation data, this power data demonstrates that RPR is an effective and efficient fault tolerant method. There is a trade-off between fault tolerance and power consumption, as expected. For applications in which slight numerical inaccuracies are acceptable or power is a significant constraint, the RPR architecture is a superior choice.

Finally, the dissertation concluded by examining RPR's performance, in terms of data quality, when applied to an image compression algorithm. The discrete cosine transform (DCT) used in many signal processing applications, such as the JPEG and MPEG standards, was implemented into a MATLAB testbed to demonstrate various SEU fault effects. For this practical example, inexact solutions due to SEUs in an RPR design cause only minor degradation in image quality. This demonstrates that in some applications occasional lower-precision results from RPR are quite acceptable, especially considering the power and area savings compared to TMR.

B. ORIGINAL CONTRIBUTIONS

The primary contribution of this dissertation is the development of RPR as a viable method of achieving fault tolerance in FPGAs. Though similar in nature to error mitigation strategies used in other fields, this research is the first work applying such

methods to FPGA circuits. It is a unique hardware fault-tolerant structure that applies concepts used in certain software designs. RPR shares some features with the typical TMR fault tolerant approach, such as spatial redundancy and voting of output data values. However, it differs fundamentally in that it recognizes that the individual output data bits from a numerical computation differ in importance. RPR protects only the most important data bits by establishing upper and lower error bounds to guard against faults in the full-precision computation module. This unique approach prevents numerically large data errors from propagating through the system.

This research shows how the RPR architecture can be applied to real-world problems to reduce the cost of hardware fault tolerance. RPR was implemented in FPGA circuits based on the well-known CORDIC algorithm. These test circuits demonstrated the advantages of RPR in terms of size and power, as well as its effectiveness in fault mitigation. Computational tasks were categorized as either Class A or Class B according to whether RPR might be appropriate in a given situation. By following the process described in the flowchart of Figure 2.12, one can quickly assess whether a particular problem is a candidate for RPR. Furthermore, it is important to stress that RPR is also quite suitable for non-FPGA systems. Though the focus in this research was on FPGA-based spacecraft computers, many of the concerns addressed here are equally valid for custom ASIC and other circuit technologies.

The second contribution of this research is the development of a total performance metric (TPM) as a means of determining the best overall solution to a given problem based on numerous performance criteria. TPM is a quantitative tool that integrates benefit factors (speed, reliability, etc.) and cost factors (power, size, etc.) into a single metric. While these benefit/cost factors and their relative weightings must be customized for each individual situation, TPM enforces a structured process for quantifying factors important to a given design. A key observation is that reliability should be considered as one of many possible important design considerations. TPM is useful for comparing various fault-tolerant techniques since it accounts for the different strengths and weaknesses of each approach. An essential feature of the TPM methodology is the flexibility that allows it to be adapted to most any design situation, not just fault-tolerant FPGA computing.

This dissertation's third main contribution is the development of the CFTP SEU simulation system. Building upon hardware and software tools developed at NPS for the CFTP program, a comprehensive method was created and validated for testing the SEU tolerance of FPGA circuit configurations. The completion of this SEU simulator is a major contribution, as it allows comprehensive ground-based testing for predicting on-orbit performance of CFTP experiments. The simulator also provides a means of replicating and studying the effects of SEUs reported by the CFTP flight experiments, once the NPSat-1 and MidSTAR-1 satellites are launched in late-2006. In addition, this SEU simulator enables detailed studies of the effectiveness of various fault-tolerant approaches, as demonstrated in this research.

Although other institutions have built similar systems for simulating SEU effects on FPGAs [8], [99], [109], this simulator is unique because it uses an actual spaceflight hardware configuration. The SEU simulators described elsewhere in the literature are based on custom setups not used directly in any space experiments. Though working with the CFTP flight hardware presents some challenges, for example limited I/O interconnectivity, it offers several advantages. First, the FPGA configuration files tested in the simulator are identical to those that can be run on CFTP during space operations. This provides the most accurate assessment of SEU sensitivities and allows ground-based verification of faults that are observed on-orbit for specific circuit configurations. Second, data from the simulator not only provides valuable reliability predictions, but also serves as excellent testing of the CFTP equipment's readiness for space operations. Finally, the capability of artificially injecting faults into the CFTP hardware offers an alternative means of exercising fault-tolerant designs during space operations. Since the expected SEU rates in the NPSat-1 and MidSTAR-1 orbits are quite low, it may be worthwhile to accelerate the CFTP experiments by using this fault injection tool.

C. FUTURE WORK

Part of the outcome from this research is recommending areas for further investigation. Several of these topics are related to better understanding and mitigating SEU effects. Additionally, some of the FPGA power consumption issues explored in this dissertation are good candidates for further exploration.

The first recommendation is to conduct more extensive radiation testing with the CFTP hardware. The testing described in Chapter VII proved useful, especially in validating the SEU simulator. However, additional testing could be used to investigate the surprisingly high SEU sensitivities discussed in Chapter VI. Although the radiation testing and simulator data matched for nearly 200 unique SEUs, only two CORDIC circuits were tested in the radiation chamber. It would useful to test the other circuits used in the simulator, as well as new TMR and RPR circuit designs that follow Xilinx's recommended I/O and voter methods. Additional testing should also include circuits that incorporate half-latch removal methods developed at LANL and BYU. As their research has shown, this is important for avoiding undetectable SEU susceptibilities.

Further radiation testing would also be useful for exploring the unexpected SEU polarity behavior found here. As discussed in Chapter VII, it is possible that in the Virtex FPGA architecture SEUs are more likely to cause 1-to-0 configuration bit upsets, whereas the Virtex-II may be more susceptible to 0-to-1 transitions. If these preliminary conclusions are confirmed, one might pursue methods of developing bitstreams with higher percentages of the less-susceptible bit polarity as a novel way of improving FPGA reliability.

Another area for future work is analyzing data from on-orbit testing with CFTP experiments. Once the NPSat-1 and MidSTAR-1 satellites are launched, the CFTP team will have access to valuable real-world SEU data. This data should be used to verify results from radiation testing and the SEU simulator. While it is expected that the spacecraft data will match ground testing, any differences found will be important for improving these predictive tools. An additional tool that would enhance these SEU studies is a means of determining the precise circuit function of each configuration bit on the FPGA. This information would enable detailed analysis of how individual SEUs cause logic errors to propagate through a circuit. Other researchers have published results based on this type of analysis, but the actual bit-to-function mappings have not been published.

With minor modifications, the SEU simulator built for this research could be used with the Virtex-II board that was used during part of the radiation testing. This would be

useful for comparing radiation data from Nov 2005 and any future tests. A Virtex-II SEU simulator would allow testing of larger and more complex circuits that might exceed the capacity of the baseline XQVR600 device on CFTP. For example, the "PIX" circuit tested at UC Davis was built under the CFTP project as a distributed-TMR experiment, but its large size meant that it could only be used on the Virtex-II prototype board.

Further theoretical work would be useful for building a stronger foundation for determining whether algorithms are Class A or Class B. While this research focused specifically on numerical computations, it would be worthwhile to determine if and when RPR can be applied to functions without an obvious hierarchy of importance in output data bits. Functions, such as Hamming distance comparators, that produce a single output bit may even be Class A if simplified approximate solutions can be developed, perhaps by ignoring portions of the input data. One might also search for the most primitive Class A function. Is the most basic reducible function a numerical operation, such as addition, or can other operations with even simpler input/output relationships be approximated?

In this research, power estimates were made using only computer simulations. Future research might involve measuring power consumption on actual FPGA devices. Accurate power measurements could verify the conclusions stated here showing that RPR used only half as much power as TMR for the CORDIC circuits tested. A major challenge of this approach is properly compensating for the power usage of ancillary components on the CFTP board. Such work might also require modifications to the CFTP circuit board(s) and/or power supply setup. Despite these challenges, power measurements would help validate the power simulation methods used in this research and improve future studies of power-efficient fault-tolerant designs.

Finally, although RPR was shown to be significantly more power efficient than TMR, additional power-saving techniques could be explored. Chapter III explains that in current FPGA technologies these techniques generally focus on the dynamic power equation. The literature contains many unique approaches to reducing power. Future work might involve applying some of these techniques to fault-tolerant designs. For

example, as Chapter VI demonstrates, pipelined CORDIC circuits use substantially less power on a per-calculation basis than the iterative CORDIC implementations.

D. CONCLUDING REMARKS

This research has culminated in a set of tools for evaluating several important characteristics of FPGA-based spacecraft computing systems. The automated SEU simulator enables exhaustive testing of a specific architecture's fault tolerance, and hence reliability, even in the preliminary stages of system development. Data from the SEU simulator can be integrated into modeling environments, such as the MATLAB testbed for the DCT image compression example, to provide visual and statistical evidence of how a system is likely to respond to SEU-induced faults within the FPGA circuits. Power estimates can be made using simulation software to assess the efficiency of various designs. The TPM method of evaluation ties these tools together by considering all important characteristics of competing designs and weighing them appropriately for a given task. The utility of this evaluation method extends well beyond the scope of the fault tolerance research conducted for this study.

Through the methodologies mentioned above, this dissertation clearly demonstrates the utility of a reduced precision redundancy architecture for fault tolerance in FPGAs and other technologies. Though much of this research focused on CORDIC test circuits, the RPR approach can easily be applied to many different FPGA and non-FPGA circuits. Many computational tasks for both orbital and terrestrial applications are amenable to "flexible precision." Flexible precision allows for area and power savings through the use of smaller, more efficient circuits. RPR takes advantage of this efficiency while providing fault protection comparable to the standard TMR method.

APPENDIX A – CORDIC PROCESSOR DESIGN

As discussed in Chapter V, a CORDIC processor was the primary test case for the fault-tolerant and power-saving techniques developed in this research. Various implementations of a basic CORDIC trigonometric calculator were constructed to provide insight into the complex interactions between a circuit's size, speed, complexity, feedback structure, power consumption and fault tolerance. This appendix describes the design methodology for producing the circuits tested and analyzed in Chapters VI and VII.

A. OVERVIEW

The basic CORDIC circuit tested follows the iterative design presented in Parhami [30] and uses simple adder/subtracter and shifter elements. Using the circular rotation mode, this circuit produces the sine and cosine of the input angle after performing a series of intermediate calculations. The fundamental CORDIC equations from Chapter V are repeated here. As explained in [30], initializing the registers with the values (0.60725, 0, <input angle>) configures the circuit for calculating cosine and sine.

$$\begin{aligned} X_{i+1} &= X_i - \xi_i Y_i 2^{-i} \\ Y_{i+1} &= Y_i + \xi_i X_i 2^{-i} \\ \lambda_{i+1} &= \lambda_i - \xi_i \tan^{-1} 2^{-i} \\ \xi_i &= \text{sign}(\lambda_i) \end{aligned} \qquad \text{with} \begin{cases} X_0 = 1/K = 0.60725 \\ Y_0 = 0 \\ \lambda_0 = \langle \text{input angle} \rangle \end{cases} \tag{A.1}$$

The baseline design was a 32-bit processor. A two's complement fixed-point number system was established to accommodate the desired range of inputs [-π/2, π/2] and outputs [-1, 1]. For input angles outside this range, a pre-rotation operation could be added as in [34], though this was not implemented in this design. Angles are represented in units of radians, though any convenient angular units could be chosen if the arctangent look-up table is modified appropriately. Using radians simplified the interpretation of results.

The circuit's inputs and outputs provide for 32-bit resolution, of which 30 bits are devoted to the fractional part. This provides sine and cosine values accurate to within $\pm 2^{-31}$, or $\pm 4.66 * 10^{-10}$. The internal calculations are performed on 39-bit operands. This wider internal datapath accounts for rounding/truncation errors. In order to achieve *n*-bit precision in the output values, the authors in [34] state that the internal wordwidth must be (n + log n + 2), although they do not provide a derivation of this formula. A derivation is found in [110], where an upper-bound error analysis was performed including both approximation and truncation errors. As mentioned above, all numbers are represented in two's complement notation and angle values are expressed in radians. The table below shows the fixed-point formats used for the baseline 32-bit CORDIC design.

Inputs & Outputs	Bit #	31	30	29	28	27	…	1	0			
	Value	-2^1	2^0	2^{-1}	2^{-2}	2^{-3}	…	2^{-29}	2^{-30}			
Internal	Bit #	38	37	36	35	34	…	8	7	…	1	0
	Value	-2^1	2^0	2^{-1}	2^{-2}	2^{-3}	…	2^{-29}	2^{-30}	…	2^{-36}	2^{-37}

Table A.1 Fixed-Point Two's Complement Number Formats for 32-Bit CORDIC

Kota's derivation [110] defines numerical precision as the number of fractional bits (i.e., number of digits to the right of the binary point). They account for the sign and 2^0 bits in the term "+ 2" in the equation above. Thus 37 bits are sufficient for 32-bit resolution. This is supported by the simulation results in [94], where precision greater than 30 fractional bits can be achieved with 36 or 37 total bits when 33 iterations are performed. Hu did not show results for a 32 iteration design as used here, but extrapolation from his tables indicates that 32 iterations are sufficient. The 39-bit design detailed above is therefore over-designed for the desired accuracy. However, since all of the radiation test data (see Chapter VII) was collected with the original 39-bit version, this internal datapath width was used for all 32-bit CORDIC circuits in this research.

B. DESIGN PROCESS

1. Schematic Design Specification

The designs used in the Aug and Nov 2005 radiation experiments were built using Xilinx ISE 6.2 and 6.3. The majority of the design was created using the schematic entry tool ECS. The ROM look-up tables were VHDL modules, while all other modules were specified schematically down to the individual logic element level (e.g., AND, OR, MUX, FF, etc.). The top-level schematic file for the 32-bit CORDIC processor is shown in Figure A.1.

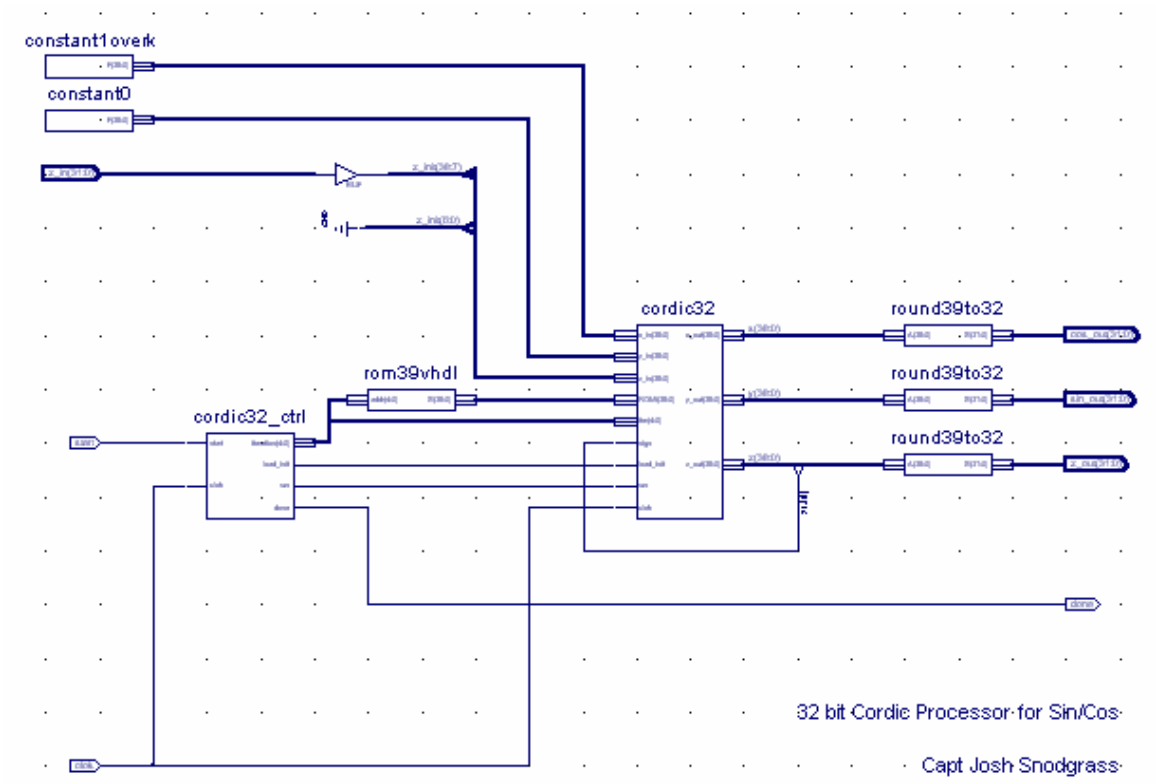


Figure A.1 32-Bit CORDIC Processor Schematic

This basic processor module was integrated into the overall FPGA design used on the CFTP board X2 experiment device. For example, in the TMR experiments this module was triplicated within X2's top-level VHDL code with simple component declarations. The top-level code included a voter unit, a counter for automatically generating input values and several control signals. As described in [103], the X1 device

provides control and interface for the experiments loaded onto X2. The basic CORDIC processor was small enough to load a copy into the X1 design for synchronous error detection of the results produced by X2. It should be noted that in this research TMR was applied only to the main data processing blocks. This follows the approach taken in [85] in which "the 'full TMR' … does not triplicate the clock, reset and I/O signals." By contrast, the methods recommended by Xilinx [68] and other researchers [8], [88] involve separate clock and I/O signals for each of the three TMR computation modules. However, these more complete TMR methods are not always feasible due to I/O and clock network limitations. For example, as discussed in Chapter VI, the CFTP board does not support I/O triplication of a 32-bit output vector.

For the Aug 2005 experiments the design was instantiated onto the CFTP development board's Xilinx Virtex XQVR600 FPGAs. For the Nov 2005 experiments a few small modifications were made to the original design and the new version was tested on both CFTP development boards, using the Virtex XQVR600 and Virtex-II XC2V6000 FPGAs. The Xilinx ISE development tools, including the XST synthesis tool, were used throughout the process. The only design constraints levied on the tools were the physical pin locations necessary to interface the circuit with the CFTP boards. Otherwise, the tools automatically optimized the design to remove redundant logic and improve the circuit's speed. No attempt was made to avoid potential "half-latch" or "keeper circuit" problems, as identified in [7], [8]. This simplified the design flow, but allowed some potential for undiagnosable faults. As pointed out in the literature, half-latches present a sizeable cross section in the Virtex devices. However, recent investigations with the Virtex-II device family show fewer problems from half-latches.

Once each circuit was thoroughly simulated and tested for functional correctness, a "golden" version of the fully placed-and-routed design was archived. Changes to the circuit are manifested as changes to the configuration bitstream, therefore accurate analysis of faults requires exact knowledge of the circuit-to-bitstream mapping. Since even the smallest design change can significantly change the placed-and-routed circuit, it was critical to not modify this golden design between radiation testing, fault injection simulations, and analysis. One of the designs tested in Aug and Nov 2005 was a TMR version of the 32-bit CORDIC. Figure A.2 below shows how the design from Aug 2005

was implemented on the Virtex device (image generated by Xilinx's FPGA Editor software). Though this design appears quite dense, it uses only 19% of the total slices on the chip.

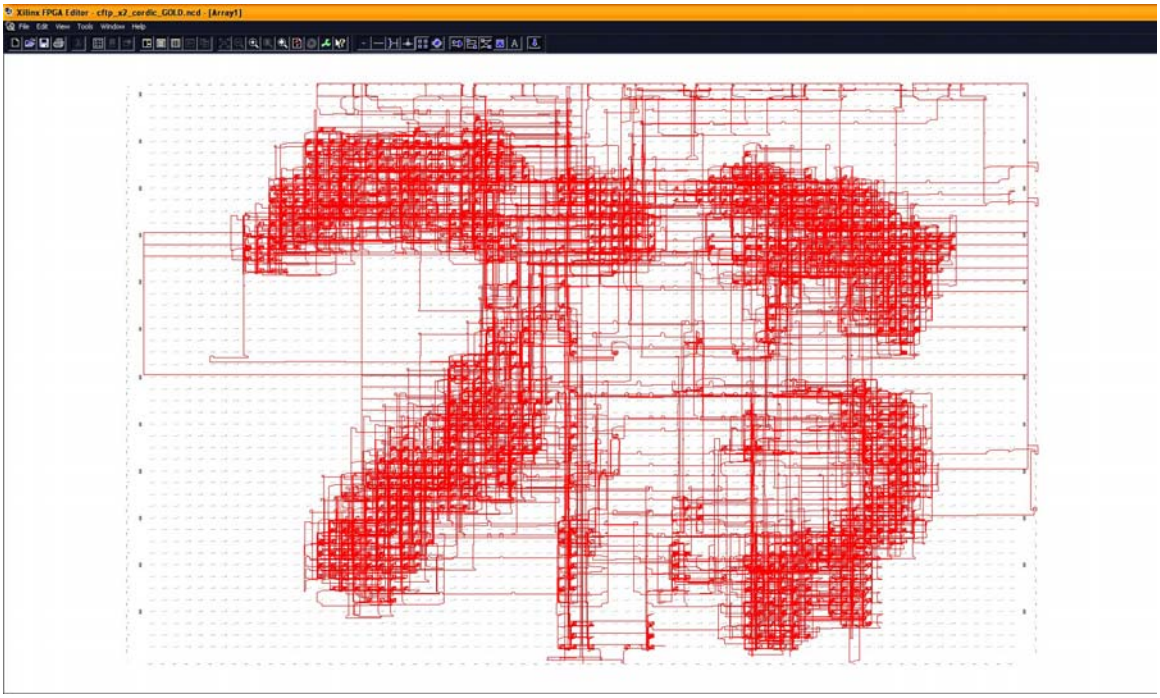


Figure A.2 Layout of TMR 32-Bit CORDIC on Virtex XQVR600

2. VHDL Design Specification for Iterative CORDIC

Following the Nov 2005 radiation tests, the design was translated into a VHDL representation. This provided more flexibility and simplified the development process. For example, changing the number of bits of precision in the schematic design requires painstaking effort, but can be rapidly implemented in VHDL with only minor modifications. Also, whereas the schematic design consisted of numerous files that had to interface correctly with one another, the VHDL design was built as a single file that could be easily ported between machines and software environments. The code listed below is functionally equivalent to the 32-bit schematic design described earlier.

```
-- Josh Snodgrass
-- 32-bit CORDIC processor
-- Iterative design:
--   32-bit precision
--   39-bit internal word length = 32 + log2(32) + 2
--   35 cycle latency = 3 start + 32 iterations (+ 2 done)

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity cordic32sin_cos_iter is
	generic (
		precision		: integer := 32;
		wordlength		: integer := 39;
		--size_diff		: integer := wordlength-precision;
		size_diff		: integer := 7;
		iter_max_cnt		: integer := 31);
	port (
		clok			: in std_logic;
		start			: in std_logic;
		z_in			: in std_logic_vector (precision-1 downto 0);
		cos_out			: out std_logic_vector (precision-1 downto 0);
		sin_out			: out std_logic_vector (precision-1 downto 0);
		ang_out			: out std_logic_vector (precision-1 downto 0);
		done			: out std_logic);
end cordic32sin_cos_iter;

architecture Behavioral of cordic32sin_cos_iter is

	signal x_int			: signed (wordlength-1 downto 0);
	signal y_int			: signed (wordlength-1 downto 0);
	signal z_int			: signed (wordlength-1 downto 0);
	signal cos_int			: signed (precision-1 downto 0);
	signal sin_int			: signed (precision-1 downto 0);
	signal ang_int			: signed (precision-1 downto 0);
	signal state				: integer range 0 to 4 := 0;
	signal iteration		: integer range 0 to iter_max_cnt;

	CONSTANT x_init   : signed (wordlength-1 downto 0) := x"26DD3B6A1" & "000";  -- 1/K
	CONSTANT y_init   : signed (wordlength-1 downto 0) := (others => '0');

	type mem_type is array (iter_max_cnt downto 0) of signed(wordlength-1 downto 0);
	signal z_LUT			: mem_type;

begin

	cos_out <= std_logic_vector(cos_int);
	sin_out <= std_logic_vector(sin_int);
	ang_out <= std_logic_vector(ang_int);

	z_LUT(0) <= "001100100100001111110110101010001000100";
	z_LUT(1) <= "000111011010110001100111000001010110000";
	z_LUT(2) <= "000011111010110110111010111111001001011";
	z_LUT(3) <= "000001111111010101101110101001101010101";
	z_LUT(4) <= "000000111111111010101011011101101110010";
	z_LUT(5) <= "000000011111111111010101010110111011101";
	z_LUT(6) <= "000000001111111111110101010101101101110";
	z_LUT(7) <= "000000000111111111111111010101010101011";
	z_LUT(8) <= "000000000011111111111111111010101010101";
	z_LUT(9) <= "000000000001111111111111111111010101010";
	z_LUT(10) <= "000000000000111111111111111111111010101";
	z_LUT(11) <= "000000000000011111111111111111111111010";
	z_LUT(12) <= "000000000000001111111111111111111111111";
	z_LUT(13) <= "000000000000000111111111111111111111111";
	z_LUT(14) <= "000000000000000011111111111111111111111";
	z_LUT(15) <= "000000000000000001111111111111111111111";
	z_LUT(16) <= "000000000000000000111111111111111111111";
```

```
z_LUT(17) <= "00000000000000000001111111111111111111";
z_LUT(18) <= "00000000000000000001111111111111111111";
z_LUT(19) <= "00000000000000000001111111111111111111";
z_LUT(20) <= "00000000000000000000111111111111111111";
z_LUT(21) <= "00000000000000000000011111111111111111";
z_LUT(22) <= "00000000000000000000001111111111111111";
z_LUT(23) <= "00000000000000000000000111111111111111";
z_LUT(24) <= "00000000000000000000000011111111111111";
z_LUT(25) <= "00000000000000000000000001111111111111";
z_LUT(26) <= "00000000000000000000000000111111111111";
z_LUT(27) <= "00000000000000000000000000010000000000";
z_LUT(28) <= "00000000000000000000000000001000000000";
z_LUT(29) <= "00000000000000000000000000000100000000";
z_LUT(30) <= "00000000000000000000000000000010000000";
z_LUT(31) <= "00000000000000000000000000000001000000";

process (clok,start) begin  -- Control signals for state machine
        if (start='1') then
                state <= 1;
        elsif (clok'event and clok='1') then
                if (state = 0) then
                        state <= 0;
                elsif (state=3) then
                        if (iteration=iter_max_cnt) then
                                state <= 4;
                        end if;
                elsif (state=4) then
                        state <= 0;
                else
                        state <= state + 1;
                end if;
        end if;
end process;

process (clok,start) begin  -- More control signals
        if (start='1') then
                iteration <= 0;
                done <= '0';
        elsif (clok'event and clok='1') then
                if (state=3) then
                        if (iteration=iter_max_cnt) then
                                iteration <= 0;
                                done <= '1';
                        else
                                iteration <= iteration + 1;
                        end if;
                end if;
        end if;
end process;

process (clok) begin  -- CORDIC computation
        if (clok'event and clok='1') then
                if (state=2) then
                        x_int <= x_init;
                        y_int <= y_init;
                        z_int <= signed(z_in) & "0000000";
                elsif (state=3) then
                        if (z_int(wordlength-1)='0') then
                                x_int <= x_int - shift_right(y_int,iteration);
                                y_int <= y_int + shift_right(x_int,iteration);
                                z_int <= z_int - z_LUT(iteration);
                        else
                                x_int <= x_int + shift_right(y_int,iteration);
                                y_int <= y_int - shift_right(x_int,iteration);
                                z_int <= z_int + z_LUT(iteration);
                        end if;
                end if;
        end if;
end process;
```

```
process (x_int,y_int,z_int) begin  -- Rounding of outputs from size "wordlength" to "precision"
        if (x_int(size_diff-1) = '1') then
                if (x_int(size_diff-2 downto 0) /= "000000") then  -- Round up
                        cos_int <= x_int(wordlength-1 downto size_diff) + 1;
                elsif (x_int(size_diff) = '1') then  -- Round to nearest even
                        cos_int <= x_int(wordlength-1 downto size_diff) + 1;
                else  -- No rounding
                        cos_int <= x_int(wordlength-1 downto size_diff);
                end if;
        else  -- No rounding
                cos_int <= x_int(wordlength-1 downto wordlength-precision);
        end if;

        if (y_int(size_diff-1) = '1') then
                if (y_int(size_diff-2 downto 0) /= "000000") then  -- Round up
                        sin_int <= y_int(wordlength-1 downto size_diff) + 1;
                elsif (y_int(size_diff) = '1') then  -- Round to nearest even
                        sin_int <= y_int(wordlength-1 downto size_diff) + 1;
                else  -- No rounding
                        sin_int <= y_int(wordlength-1 downto size_diff);
                end if;
        else  -- No rounding
                sin_int <= y_int(wordlength-1 downto wordlength-precision);
        end if;

        if (z_int(size_diff-1) = '1') then
                if (z_int(size_diff-2 downto 0) /= "000000") then  -- Round up
                        ang_int <= z_int(wordlength-1 downto size_diff) + 1;
                elsif (z_int(size_diff) = '1') then  -- Round to nearest even
                        ang_int <= z_int(wordlength-1 downto size_diff) + 1;
                else  -- No rounding
                        ang_int <= z_int(wordlength-1 downto size_diff);
                end if;
        else  -- No rounding
                ang_int <= z_int(wordlength-1 downto wordlength-precision);
        end if;
end process;

end Behavioral;
```

3. VHDL Design Specification for Pipelined CORDIC

In addition, a pipelined version of the CORDIC was built as a VHDL design for power simulations in Chapter VI. The code listed below provides 32 bits of precision with 32 clock cycles of latency, but throughput is dramatically improved since results are produced on every clock cycle. Of course this gain in speed requires increased circuit area. With this 32-bit design, the Xilinx ISE tools yield a circuit that occupies 2131 slices (30% of total slices) on the XQVR600 part.

```
-- Josh Snodgrass
-- 32 bit CORDIC processor
-- Pipeline design:
--   32 bit precision
--   39 bit internal word length = 32 + log_2(32) + 2
--   32 cycle latency

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
```

```
entity cordic32sin_cos_pipe is
	generic (
		precision		: integer := 32;
		wordlength		: integer := 39;
		--size_diff		: integer := wordlength-precision;
		size_diff		: integer := 7);
	port (
		clok			: in std_logic;
		z_in			: in std_logic_vector (precision-1 downto 0);
		cos_out			: out std_logic_vector (precision-1 downto 0);
		sin_out			: out std_logic_vector (precision-1 downto 0);
		ang_out			: out std_logic_vector (precision-1 downto 0));
end cordic32sin_cos_pipe;

architecture Behavioral of cordic32sin_cos_pipe is

	signal cos_int			: signed (precision-1 downto 0);
	signal sin_int			: signed (precision-1 downto 0);
	signal ang_int			: signed (precision-1 downto 0);

	CONSTANT x_init			: signed (wordlength-1 downto 0) := x"26DD3B6A1" & "000";  -- 1/K
	CONSTANT y_init			: signed (wordlength-1 downto 0) := (others => '0');

	type reg_type is array (0 to precision) of signed(wordlength-1 downto 0);
	signal x_int			: reg_type;
	signal y_int			: reg_type;
	signal z_int			: reg_type;

	type mem_type is array (0 to precision-1) of signed(wordlength-1 downto 0);
	signal z_LUT			: mem_type;

begin

	cos_out <= std_logic_vector(cos_int);
	sin_out <= std_logic_vector(sin_int);
	ang_out <= std_logic_vector(ang_int);

	z_LUT(0) <= "0011001001000011111101101010100010001000100";
	z_LUT(1) <= "0001110110101100011001110000010101100000";
	z_LUT(2) <= "0000111110101101101110101111111001001011";
	z_LUT(3) <= "0000011111110101011011101010011101010101";
	z_LUT(4) <= "0000001111111110101010110111011101100010";
	z_LUT(5) <= "0000000111111111111010101010110111011101";
	z_LUT(6) <= "0000000011111111111111010101010101101110";
	z_LUT(7) <= "0000000001111111111111110101010101011";
	z_LUT(8) <= "0000000000111111111111111110101010101";
	z_LUT(9) <= "0000000000011111111111111111111010101010";
	z_LUT(10) <= "0000000000001111111111111111111111010101";
	z_LUT(11) <= "0000000000000111111111111111111111111010";
	z_LUT(12) <= "0000000000000011111111111111111111111111";
	z_LUT(13) <= "0000000000000001111111111111111111111111";
	z_LUT(14) <= "0000000000000000111111111111111111111111";
	z_LUT(15) <= "0000000000000000011111111111111111111111";
	z_LUT(16) <= "0000000000000000001111111111111111111111";
	z_LUT(17) <= "0000000000000000000111111111111111111111";
	z_LUT(18) <= "0000000000000000000011111111111111111111";
	z_LUT(19) <= "0000000000000000000001111111111111111111";
	z_LUT(20) <= "0000000000000000000000111111111111111111";
	z_LUT(21) <= "0000000000000000000000011111111111111111";
	z_LUT(22) <= "0000000000000000000000001111111111111111";
	z_LUT(23) <= "0000000000000000000000000111111111111111";
	z_LUT(24) <= "0000000000000000000000000011111111111111";
	z_LUT(25) <= "0000000000000000000000000001111111111111";
	z_LUT(26) <= "0000000000000000000000000000111111111111";
	z_LUT(27) <= "0000000000000000000000000000010000000000";
	z_LUT(28) <= "0000000000000000000000000000001000000000";
	z_LUT(29) <= "0000000000000000000000000000000100000000";
	z_LUT(30) <= "0000000000000000000000000000000010000000";
	z_LUT(31) <= "0000000000000000000000000000000001000000";
```

```
	process (clok) begin  -- CORDIC computation
		if (clok'event and clok='1') then
			x_int(0) <= x_init;
			y_int(0) <= y_init;
			z_int(0) <= signed(z_in) & "0000000";

			for i in 0 to precision-1 loop
				if (z_int(i)(wordlength-1)='0') then
					x_int(i+1) <= x_int(i) - shift_right(y_int(i),i);
					y_int(i+1) <= y_int(i) + shift_right(x_int(i),i);
					z_int(i+1) <= z_int(i) - z_LUT(i);
				else
					x_int(i+1) <= x_int(i) + shift_right(y_int(i),i);
					y_int(i+1) <= y_int(i) - shift_right(x_int(i),i);
					z_int(i+1) <= z_int(i) + z_LUT(i);
				end if;
			end loop;
		end if;
	end process;

	process (x_int(precision),y_int(precision),z_int(precision)) begin
		-- Rounding of outputs from size "wordlength" to "precision"

		if (x_int(precision)(size_diff-1) = '1') then
			if (x_int(precision)(size_diff-2 downto 0) /= "000000") then  -- Round up
				cos_int <= x_int(precision)(wordlength-1 downto size_diff) + 1;
			elsif (x_int(precision)(size_diff) = '1') then  -- Round to nearest even
				cos_int <= x_int(precision)(wordlength-1 downto size_diff) + 1;
			else  -- No rounding
				cos_int <= x_int(precision)(wordlength-1 downto size_diff);
			end if;
		else  -- No rounding
			cos_int <= x_int(precision)(wordlength-1 downto wordlength-precision);
		end if;

		if (y_int(precision)(size_diff-1) = '1') then
			if (y_int(precision)(size_diff-2 downto 0) /= "000000") then  -- Round up
				sin_int <= y_int(precision)(wordlength-1 downto size_diff) + 1;
			elsif (y_int(precision)(size_diff) = '1') then  -- Round to nearest even
				sin_int <= y_int(precision)(wordlength-1 downto size_diff) + 1;
			else  -- No rounding
				sin_int <= y_int(precision)(wordlength-1 downto size_diff);
			end if;
		else  -- No rounding
			sin_int <= y_int(precision)(wordlength-1 downto wordlength-precision);
		end if;

		if (z_int(precision)(size_diff-1) = '1') then
			if (z_int(precision)(size_diff-2 downto 0) /= "000000") then  -- Round up
				ang_int <= z_int(precision)(wordlength-1 downto size_diff) + 1;
			elsif (z_int(precision)(size_diff) = '1') then  -- Round to nearest even
				ang_int <= z_int(precision)(wordlength-1 downto size_diff) + 1;
			else  -- No rounding
				ang_int <= z_int(precision)(wordlength-1 downto size_diff);
			end if;
		else  -- No rounding
			ang_int <= z_int(precision)(wordlength-1 downto wordlength-precision);
		end if;
	end process;

end Behavioral;
```

C. SYNTHESIS TOOL VARIABILITY

In addition to functional correctness, performance factors such as speed and area are important when compiling designs for actual FPGA implementation. Such performance factors are strongly dependant on the exact design, but other issues related to the design synthesis environment also affect performance. The tools that translate a schematic or VHDL design into an FPGA configuration file are extremely complicated. Several different commercial vendors offer competing toolsets. Within a given vendor product line, there are typically several levels of design tools and various generations of software versions. Moving between these different design environments can considerably influence the final placed-and-routed design. Furthermore, there are a tremendous number of design optimization choices available within a particular design package. Arriving at an optimal balance between speed, area, power, etc. is complicated by the numerous choices regarding software tools and optimization settings. To achieve consistent and predictable designs requires careful management of the source files describing the design as well as following a repeatable design synthesis process.

The CFTP team established Xilinx's ISE (Integrated Software Environment) version 6.2i for LINUX as the baseline development environment. This toolset includes XST (Xilinx Synthesis Tool), PAR (Place-And-Route) and other circuit mapping and miscellaneous programs. The company Synplicity offers a competing package featuring their Synplify design synthesis software. Synplify is widely used in industry and considered superior to XST. Although it was not the intent of this research to produce highly optimized CORDIC circuits, it is interesting to compare the results from these competing software packages. Table A.2 shows how well XST and Synplify compiled the CORDIC designs for several different levels of numerical precision.

Design Method	External / Internal Wordlength [bits]	Slice Count (% of chip)		Max Delay [nsec]	
		XST	Synplify	XST (ModelSim)	Synplify
Schematic	32 / 39	472 (6%)	-	69 (40)	-
VHDL iterative	32 / 39	892 (12%)	373 (5%)	26 (30)	22
	16 / 20	265 (3%)	177 (2%)	20 (24)	20
	8 / 11	124 (1%)	83 (1%)	16 (22)	16
VHDL pipeline	32 / 39	2131 (30%)	1931 (27%)	17 (26)	17
	16 / 20	554 (7%)	459 (6%)	12 (22)	13
	8 / 11	165 (2%)	136 (1%)	11 (20)	14

Table A.2 CORDIC Circuit Sizes and Speeds on Virtex XQVR600

In every case, Synplify yielded considerably smaller circuits with comparable delay to those produced by XST. Circuit size should scale approximately linearly with wordlength. For the iterative circuits a doubling in wordlength should cause a doubling in size. This trend is followed in the Synplify data. However, XST was particularly inefficient when compiling the "VHDL iterative: 32 / 39" design, using more than twice as many slices as Synplify for the same design. Note that the functionally equivalent 32-bit schematic design compiled through XST had slice usage much closer to the VHDL version via Synplify. Furthermore, the XST versions of the 16-bit and 8-bit iterative circuits differ by a factor of two, so there was clearly some anomalous behavior in the 32-bit iterative XST design. For the pipeline circuits, a doubling in wordlength should cause a quadrupling in size. Both XST and Synplify follow this trend fairly well.

Also shown in the table are estimated timing delays for the placed and routed circuits. High-fidelity circuit simulations, such as ModelSim, can provide more accurate timing estimates. ModelSim was used to verify the timing estimates from the synthesis tools. These tests were not exhaustive since testing all 2^{32} possible input values would be prohibitively time-consuming. However, several sample input values were run through each circuit so the delay between the clock edge and valid output data could be measured. Timing delays reported by the synthesis tools were consistently lower that the delays observed in the simulator. One notable exception was the 32-bit schematic version. Note

from the table above that the estimated delay for the schematic design is over twice that for the functionally equivalent VHDL version. However, running several input values through the simulator yielded a maximum delay of only 40 nsec, compared to the estimated max delay of 69 nsec. To better understand this discrepancy, a design was built that contained one copy of the schematic version and one copy of the VHDL version. This design was run through the simulator and several dozen computation sequences were analyzed. Surprisingly, the VHDL version had slightly larger delays for most of the input values analyzed, though its maximum delay was less. This data demonstrates that one must be cautious when interpreting timing estimates and, in general, one should use more conservative (i.e., slower) clock speeds than indicated from the estimates.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B – MATLAB ERROR SIMULATION CODE

This appendix contains the MATLAB code used in Chapters V and VIII for simulating faults in CORDIC and DCT calculations.

A. CORDIC ERROR PROPAGATION CODE

1. CORDIC Algorithm with Specific Forced Errors

```
% Josh Snodgrass
% CORDIC simulation for Chapter 5
%
% THIS VERSION INDUCES SPECIFIC ERRORS!!!!!
%
% Calculation of sine(z) and cosine(z)
%
% Current Limitations:
% input range -pi <= z <= +pi

clear all; close all;

%Calculate K for m iterations
m=8;  % input/output wordlength, with 2 bits for sign and 1's
place
n=(m-2)+ceil(log2(m-2));  % internal precision, # fractional bits
K=1;
for j=0:m-1
  K=K*sqrt(1+2^(-2*j));
end

xin=1/K;
yin=0;
zin=pi/6; % Input angle = 30 degrees
ein=atan(2.^[0:-1:-(m-1)]);

%Pre-rotation
quadrant2=zin>pi/2;
quadrant3=zin<-pi/2;
if quadrant2
  zin=zin-pi;
elseif quadrant3
  zin=zin+pi;
end

[tmp,x0]=dec2bin(xin,n);
x0=[0 0 x0];
[tmp,y0]=dec2bin(yin,n);
y0=[0 0 y0];
[z0w,z0v]=dec2bin(abs(zin),n);
z0=[0 0 z0v];
```

```
if z0w(length(z0w))
  z0(2)=1;
end
if zin<0;
  z0=addbin2(~z0,zeros(size(z0)),1);
end
for i=1:length(ein)
  [tmp,e(i,:)]=dec2bin(ein(i),n);
end
e=[zeros(size(e,1),2) e];

x(1,:)=x0;
y(1,:)=y0;
z(1,:)=z0;

for i=1:m

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Inject error in Y register during iteration 6
%  if i==6
%    y(i,4)=~y(i,4);
%  end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  d(i)=~z(i,1);
%  x(i+1)=x(i)-d(i)*y(i)/2^(i-1);
  if y(i,1)==1
    yshift=[ones(1,i-1) y(i,1:size(y,2)-(i-1))];
  else
    yshift=[zeros(1,i-1) y(i,1:size(y,2)-(i-1))];
  end
  if d(i)
    yshift=comp2s(yshift);
  end
  x(i+1,:)=addbin2(x(i,:),yshift,0);

%  y(i+1)=y(i)+d(i)*x(i)/2^(i-1);
  if x(i,1)==1
    xshift=[ones(1,i-1) x(i,1:size(x,2)-(i-1))];
  else
    xshift=[zeros(1,i-1) x(i,1:size(x,2)-(i-1))];
  end
  if ~d(i)
    xshift=comp2s(xshift);
  end
  y(i+1,:)=addbin2(y(i,:),xshift,0);

%  z(i+1)=z(i)-d(i)*e(i);
  evalue=e(i,:);
  if d(i)
    evalue=comp2s(evalue);
  end
  z(i+1,:)=addbin2(z(i,:),evalue,0);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Inject error in stage 3 z-adder
%  if i==3
%    z(i+1,:)=addbin3(z(i,:),evalue,0);
```

```
%  end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

end

%Post-rotation
if quadrant2|quadrant3
  xfinal=comp2s(x(size(x,1),:));
  yfinal=comp2s(y(size(y,1),:));
else
  xfinal=x(size(x,1),:);
  yfinal=y(size(y,1),:);
end

%Display outputs
if xfinal(1)
  xout=comp2s(xfinal);
  xout=-bin2dec(xout(2),xout(3:length(xout)));
else
  xout=bin2dec(xfinal(2),xfinal(3:length(xfinal)));
end

if yfinal(1)
  yout=comp2s(yfinal);
  yout=-bin2dec(yout(2),yout(3:length(yout)));
else
  yout=bin2dec(yfinal(2),yfinal(3:length(yfinal)));
end

zfinal=z(m+1,:);
if zfinal(1)
  zout=comp2s(zfinal);
  zout=-bin2dec(zout(2),zout(3:length(zout)));
else
  zout=bin2dec(zfinal(2),zfinal(3:length(zfinal)));
end
```

2. Supporting MATLAB Code for CORDIC Calculations

```
function sum = addbin3(x,y,cin)
% Josh Snodgrass
% Binary vector addition, ignore carry out
%
% Useful for 2's complement addition
% Inputs "x" and "y" are vectors
% Input "cin" is a scalar
%
% Input vector lengths for "x" and "y" must be equal
% Resultant vector "sum" has length = length(x)
%
% THIS VERSION INDUCES AN ERROR IN CARRY LOGIC!!!!!
```

```
if nargin<3
  disp('addbin3 accepts up to 3 inputs: x, y, cin');
end

if length(x)~=length(y)
  error('Input vectors must have same length');
end

c=zeros(1,length(x)+1);
if exist('cin')
  c(length(c))=cin;
end

for i=length(x):-1:1
  sum(i)=xor(xor(x(i),y(i)),c(i+1));
  c(i)=(x(i)&y(i))|(x(i)&c(i+1))|(y(i)&c(i+1));
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Inject error into position #6 carry bit
  if i==6
    c(i)=~c(i);
  end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
end


function [w,v] = dec2bin(x,n)
% Josh Snodgrass
% Conversion of decimal number to binary representation
%  To see both whole and fractional parts of number, use
% [w,v]=dec2bin(x,n)
%
% Current Limitations:
% input x must be non-negative
% output v set to n bits

if x<0
  error('Input value must be non-negative');
end

xw=fix(x);
xv=rem(x,1);

w=0;
i=1;
while(xw>0)
  w(i)=rem(xw,2);
  xw=fix(xw/2);
  i=i+1;
end
w=fliplr(w);

i=1;
while(i<=n)
  v(i)=fix(xv*2);
  xv=rem(xv*2,1);
  i=i+1;
end
```

```
if xv>=0.5 % round up
    xtemp=[0 w v];
    xtemp=addbin2(xtemp,zeros(length(xtemp)),1);
    w=xtemp(1:length(xtemp)-n);
    v=xtemp(length(xtemp)-n+1:length(xtemp));
end


function x = bin2dec(w,v)
% Josh Snodgrass
% Conversion of binary number to decimal representation
%  To see both whole and fractional parts of number, use
% x=bin2dec(w,v)
%
% Current Limitations:
% input [w.v] must be non-negative

x=0;

xw=fix(x);
xv=rem(x,1);

for i=1:length(w)
  x=x+w(length(w)-(i-1))*2^(i-1);
end

for i=1:length(v)
  x=x+v(i)*2^(-1*i);
end


function w = comp2s(x)
% Josh Snodgrass
% Binary 2's complement --> first digit of w and x indicates sign
% Does check for allowable range, since 2's complement
% has larger range for negative numbers vs positive numbers
%
% Current limitations:
% Assumes x is a row vector of binary values {0,1}

if length(x)<2
  error('Input binary vector must have at least 2 elements!');
end

if x(1) & sum(x(2:length(x)))==0
  error('Input binary vector is at max negative range, complement
too big!');
end

w=addbin2(~x,zeros(size(x)),1);
```

B. DCT ERROR SIMULATION CODE

```
% Josh Snodgrass
%
% Error simulation code for DCT Algorithm

clear all; close all;

filename='goldhill.bmp'; filefrmt='bmp';
[g,cmap]=imread(filename,filefrmt);

[M,N]=size(g);
g=double(g);
G=zeros(size(g));
Gfaultfree=zeros(size(g));

block=8;

err_duration=18350/(block^2);
      % 70 msec reconfig * 512x512 pixels / 1 Hz
%err_duration=8/(block^2);
      % 24 microsec partial * 512x512 pixels / 1 Hz
err_start_x=floor((N/block)*rand);
err_start_y=floor((M/block)*rand);
err_stop_x=err_start_x-1+…
      ceil((err_duration+err_start_y)/(M/block));
err_stop_y=mod(err_start_y+err_duration,(M/block));

for x=1:block:N
    for y=1:block:M
        err=0;
        if ( floor(x/block)==err_start_x )
            if ( floor(x/block)==err_stop_x )
                if ( floor(y/block)>=err_start_y & …
                        floor(y/block)<=err_stop_y )
                    err=1;
                end
            elseif ( floor(y/block)>=err_start_y )
                err=1;
            end
        elseif ( floor(x/block)>err_start_x & …
                  floor(x/block)<err_stop_x)
            err=1;
        elseif ( floor(x/block)==err_stop_x & …
                  floor(y/block)<=err_stop_y)
            err=1;
        end

        Gfaultfree(x:x+block-1,y:y+block-1)=…
            DCT_josh(g(x:x+block-1,y:y+block-1),block,0);
        G(x:x+block-1,y:y+block-1)=…
            DCT_josh(g(x:x+block-1,y:y+block-1),block,err);
    end
end
```

```
for x=1:block:N
    for y=1:block:M
        h(x:x+block-1,y:y+block-1)=…
            IDCT_josh(G(x:x+block-1,y:y+block-1),block);
    end
end

g_h=g-h;
MSE=(1/(M*N))*(sum(sum(g_h.^2))),
PSNR=10*log10((255^2)/MSE),

figure(1);
image(g); set(gcf,'Colormap',cmap,'Position',[25 525 500 450]);
axis equal;
title('Original Image');

figure(3);
image(h); set(gcf,'Colormap',cmap,'Position',[550 525 500 450]);
axis equal;
title('Restored Image');


function G = DCT_josh(g,N,err)
% Josh Snodgrass
%
% DCT Algorithm (square matrix)
% G(u,v) = (2/N)*c(u)*c(v)*(sum(sum(g(m,n)*
%          cos((2*m+1)*pi*u/(2*N))*cos((2*n+1)*pi*v/(2*N));
%  where
%   u,v=0,1,2,...,N-1
%   c(0)=1/sqrt(2)
%   c(n)=1

c=ones(1,N); c(1)=1/sqrt(2);

maxG=N*255; % for 8-bit source images

for u=1:N
    for v=1:N
        Gscale=(2/N)*c(u)*c(v);
        Gsum=0;
        for m=1:N
            for n=1:N
                Gsum=Gsum+g(m,n)*cos((2*m-1)*pi*…
                  (u-1)/(2*N))*cos((2*n-1)*pi*(v-1)/(2*N));
            end
        end
        %G(u,v)=Gscale*Gsum;
        % Rounding result to 32-bit fixed point format
        % 8-bit source image --> DCT coefficients +/- 2040
        % 16-bit source image --> DCT coefficients +/- 524,288
        G(u,v)=(fix((10^6)*Gscale*Gsum))/(10^6);
            % 32-bit --> steps = 10^-6
        %G(u,v)=128*(fix(Gscale*Gsum/128));
            % 5-bit --> steps = 128
    end
end
```

```
if err
    Gpower=sum(sum(abs(G)));
    for u=1:N
        for v=1:N
            G(u,v)=2*(rand-0.5)*maxG;
        end
    end
    Gpower2=sum(sum(abs(G)));
    G=G./(Gpower2/Gpower);
end

if 0 %err
    for u=1:N
        for v=1:N
            G(u,v)=(fix(0.0629*G(u,v))/0.0629);
                  % 8-bit --> steps = 15.9
            %G(u,v)=(fix(0.00784*G(u,v)))/0.00784;
                  % 5-bit --> steps = 127.5
        end
    end
end


function g = IDCT_josh(G,N)
% Josh Snodgrass
%
% Inverse-DCT Algorithm (square matrix)
% g(m,n) = (2/N)*(sum(sum(c(u)*c(v)*G(u,v)*
%          cos((2*m+1)*pi*u/(2*N))*cos((2*n+1)*pi*v/(2*N));
%  where
%   m,n=0,1,2,...,N-1
%   c(0)=1/sqrt(2)
%   c(n)=1

c=ones(1,N); c(1)=1/sqrt(2);

for m=1:N
    for n=1:N
        gscale=(2/N);
        gsum=0;
        for u=1:N
            for v=1:N
                gsum=gsum+c(u)*c(v)*G(u,v)*cos((2*m-1)*pi*
                  (u-1)/(2*N))*cos((2*n-1)*pi*(v-1)/(2*N));
            end
        end
        g(m,n)=gscale*gsum;
    end
end
```

LIST OF REFERENCES

[1] M. Wirthlin, E. Johnson, N. Rollins, M. Caffrey, and P. Graham, "The Reliability of FPGA Circuit Designs in the Presence of Radiation Induced Configuration Upsets," in *Proc. 11th Annual IEEE Symp. on Field-Programmable Custom Computing Machines*, Apr 2003

[2] P. Graham, M. Caffrey, M. Wirthlin, D. E. Johnson, and N. Rollins, "Reconfigurable Computing in Space: From Current Technology to Reconfigurable Systems-on-a-Chip," in *Proc. IEEE Aerospace Conf.*, Mar 2003

[3] D. Brodrick, A. Dawood, N. Bergmann, and M. Wark, "Error Detection for Adaptive Computing Architectures in Spacecraft Applications," in *Proc. 6th Australasian Computer Systems Architecture Conf.*, Jan 2001

[4] P. Graham, M. Caffrey, J. Zimmerman, P. Sundararajan, E. Johnson, and C. Patterson, "Consequences and Categories of SRAM FPGA Configuration SEUs," in *Proc. 6th Annual Conf. on MAPLD*, Sep 2003

[5] F. Sturesson, S. Mattsson, C. Carmichael, and R. Harboe-Sorensen, "Heavy Ion Characterization of SEU Mitigation Methods for the Virtex FPGA," in *Proc. 6th European Conf. on Radiation and Its Effects on Components and Systems*, Sep 2001

[6] C. C. Yui, G. M. Swift, C. Carmichael, R. Koga, and J. S. George, "SEU Mitigation Testing of Xilinx Virtex II FPGAs," *IEEE Radiation Effects Data Workshop*, Jul 2003

[7] P. Graham, M. Caffrey, D. E. Johnson, N. Rollins, and M. Wirthlin, "SEU Mitigation for Half-Latches in Xilinx Virtex FPGAs," *IEEE Trans. Nuclear Science*, Dec 2003

[8] F. Lima, C. Carmichael, J. Fabula, R. Padovani, and R. Reis, "A Fault Injection Analysis of Virtex FPGA TMR Design Methodology," in *Proc. 6th European Conf. on Radiation and Its Effects on Components and Systems*, Sep 2001

[9] N. Rollins, M. J. Wirthlin, and P. Graham, "Evaluation of Power Costs in Applying TMR to FPGA Designs," in *Proc. 7th Annual Conf. on MAPLD*, Sep 2004

[10] J. H. Anderson and F. N. Najm, "Power Estimation Techniques for FPGAs," *IEEE Trans. VLSI Systems*, Oct 2004

[11] J. Shea, "Computer Design Problems for the Space Environment," in *Proc. Conf. on Spaceborne Computer Engineering*, Oct 1962

[12] G. W. Donohoe and P. S. Yeh, "Low-Power Reconfigurable Processor," in *Proc. IEEE Aerospace Conf.*, Mar 2002

[13] E. P. Stabler and C. J. Creveling, "Spacecraft Computers for Scientific Information Systems," in *Proceedings of the IEEE*, Dec 1966

[14] P. L. Murray and D. VanBuren, "Single Event Effect Mitigation in ReConfigurable Computers for Space Applications," in *Proc. IEEE Aerospace Conf.*, Mar 2005

[15] S. C. Persyn, M. McLelland, M. Epperly, and B. Walls, "Evolution of Digital Signal Processing Based Spacecraft Computing Solutions," in *Proc. IEEE Aerospace Conf.*, Mar 2002

[16] D. Roussel-Dupre, M. Caffrey, J. Buckley, and P. Davies, "Cibola Flight Experiment," in *Proc. AIAA/USU Conf. on Small Satellites*, Aug 2004

[17] G. Messenger and M. Ash, *Single Event Phenomena*, Chapman and Hall, 1997

[18] P. P. Shirvani, "Fault-Tolerant Computing for Radiation Environments," PhD Dissertation, Stanford University, 2001

[19] K. Clark, "Modeling SET's in Complex Digital Systems," PhD Dissertation, Naval Postgraduate School, 2002

[20] W. Gibbons and H. Ames, "Use of FPGAs in Critical Space Flight Applications – A Hard Lesson," in *Proc. 2nd Annual Conf. on MAPLD*, Sep 1999

[21] K. Mohanram, "Lowering Power Consumption in Concurrent Checkers via Input Ordering," *IEEE Trans. VLSI Systems*, Nov 2004

[22] H. Quinn and P. Graham, "Terrestrial-based Radiation Upsets: a Cautionary Tale," in *Proc. 13th Annual IEEE Symp. on Field-Programmable Custom Computing Machines*, Apr 2005

[23] J. F. Wakerly, *Digital Design: Principles and Practices*, 2nd ed, Prentice Hall, 1994

[24] Xilinx Corp., "Important Dates in Xilinx History," (website) http://www.xilinx.com/company/xilinxstory/timeline.htm, Aug 2005

[25] U. Meyer-Baese, *DSP with FPGAs*, Springer, 2001

[26] D. J. Mabry, S. J. Hansel, and J. B. Blake, "The SAMPEX Data Processing Unit," *IEEE Trans. Geoscience and Remote Sensing*, May 1993

[27] J. Oberg, "Titan Calling," *IEEE Spectrum*, Oct 2004

[28] S. Y. Yu, "Fault Tolerance in Adaptive Real-Time Computing Systems," PhD Dissertation, Stanford University, 2001

[29] A. Maheshwari, W. Burleson, and R. Tessier, "Trading Off Transient Fault Tolerance and Power Consumption in Deep Submicron (DSM) VLSI Circuits," *IEEE Trans. VLSI Systems*, Mar 2004

[30] B. Parhami, *Computer Arithmetic*, Oxford University Press, 2000

[31] P. Ranganathan, "Power Management – Guest Lecture for CS4135, NPS," Naval Postgraduate School, Nov 2004

[32] R. L. Phelps, "Operational Experiences with the Petite Amateur Navy Satellite – PANSAT," in *Proc. 15th Annual AIAA/USU Conf. on Small Satellites*, Aug 2001

[33] M. Caffrey, "A Space-Based Reconfigurable Radio," in *Proc. 5th Annual Conf. on MAPLD*, Sep 2002

[34] S. Vadlamani and W. Mahmoud, "Comparison of CORDIC Algorithm Implementations on FGPA families," in *Proc. 34th Southeastern Symp. on System Theory*, Mar 2002

[35] W. J. Huang, N. Saxena, and E. J. McCluskey, "A Reliable LZ Data Compressor on Reconfigurable Coprocessors," in *Proc. Symp. on Field-Programmable Custom Computing Machines*, Apr 2000

[36] S. Hong, S. S. Chin, and D. Connaway, "Variable-Rate Pipelined Multiplier Design for Reconfigurable DSP Applications," in *Proc. 45th Midwest Symp. on Circuits and Systems*, Aug 2002

[37] R. V. K. Pillai, S. Y. A. Shah, A. J. Al-Khalili, and D. Al-Khalili, "Low Power Floating Point MAFs – A Comparative Study," in *Proc. 6th Intl. Symp. on Signal Processing and its Applications*, Aug 2001

[38] C. Nagendra, M. J. Irwin, and R. M. Owens, "Area-Time-Power Tradeoffs in Parallel Adders," *IEEE Trans. Circuits and Systems II: Analog and Digital Signal Processing*, Oct 1996

[39] T. R. N. Rao, *Error Coding for Arithmetic Processors*, Academic Press, 1974

[40] D. P. Siewiorek and R. S. Swarz, *The Theory and Practice of Reliable System Design*, Digital Press, 1982

[41] F. Lima, L. Carro, and R. Reis, "Reducing Pin and Area Overhead in Fault-Tolerant FPGA-based Designs," in *Proc. ACM/SIGDA 11th Intl. Symp. on FPGAs*, Feb 2003

[42] D. E. Johnson, K. S. Morgan, M. J. Wirthlin, M. P. Caffrey, and P. S. Graham, "Detection of Configuration Memory Upsets Causing Persistent Errors in SRAM-based FPGAs," in *Proc. 7th Annual Conf. on MAPLD*, Sep 2004

[43] M. Caffrey, P. Graham, E. Johnson, M. Wirthlin, N. Rollins, and C. Carmichael, "Single-Event Upsets in SRAM FPGAs," in *Proc. 5th Annual Conf. on MAPLD*, Sep 2002

[44] M. Ceschia, M. Violante, M. Sonza Reorda, A. Paccagnella, P. Bernardi, M. Rebaudengo, D. Bortolano, M. Bellato, P. Zambolin, and A. Candelori, "Identification and Classification of Single-Event Upsets in the Configuration Memory of SRAM-Based FPGAs," *IEEE Trans. Nuclear Science*, Dec 2003

[45] A. Doumar and H. Ito, "Detecting, Diagnosing, and Tolerating Faults in SRAM-based Field Programmable Gate Arrays: a Survey," *IEEE Trans. VLSI Systems*, Jun 2003

[46] S. Mitra and E. J. McCluskey, "Which Concurrent Error Detection Scheme to Choose?" in *Proc. International Test Conference*, Oct 2000

[47] T. R. Stankovic, M. K. Stojcev, and G. L. Djordjevic, "Design of Self-Checking Combinational Circuits," in *Proc. 6th Intl. Conf. on Telecommunications in Modern Satellite, Cable and Broadcasting Service (TELSIKS)*, Oct 2003

[48] G. M. Swift, S. Rezgui, J. George, C. Carmichael, M. Napier, J. Maksymowicz, J. Moore, A. Lesea, R. Koga, and T. F. Wrobel, "Dynamic Testing of Xilinx Virtex-II Field Programmable Gate Array (FPGA) Input/Output Blocks (IOBs)," *IEEE Trans. Nuclear Science*, Dec 2004

[49] Xilinx Corp., "Application Note 216: Correcting Single-Event Upsets Through Virtex Partial Configuration," version 1.0, Jun 2000

[50] J. DeVale, "Traditional Reliability," (website) http://www.ece.cmu.edu/~koopman/des_s99/traditional_reliability/, Sep 2005

[51] P. K. Samudrala, J. Ramos, and S. Katkoori, "Selective Triple Modular Redundancy (STMR) Based Single-Event Upset (SEU) Tolerant Synthesis for FPGAs," *IEEE Trans. Nuclear Science*, Oct 2004

[52] V. Chandrasekhar, S. N. Mahammad, V. Muralidaran, and V. Kamakoti, "Reduced Triple Modular Redundancy for Tolerating SEUs in SRAM-based FPGAs," in *Proc. 8th Annual Conf. on MAPLD*, Sep 2005

[53] B. Pratt, M. Caffrey, P. Graham, K. Morgan, and M. Wirthlin, "Improving FPGA Design Robustness with Partial TMR," in *Proc. 8th Annual Conf. on MAPLD*, Sep 2005

[54] T. R. N. Rao, "Biresidue Error-Correcting Codes for Computer Arithmetic," *IEEE Trans. Computers*, May 1970

[55] B. Shim, S. R. Sridhara, and N. R. Shanbhag, "Reliable Low-Power Digital Signal Processing via Reduced Precision Redundancy," in *IEEE Trans. VLSI Systems*, May 2004

[56] B. Littlewood, "The Use of Proof in Diversity Arguments," *IEEE Trans. Software Engineering*, Oct 2000

[57] M. Caccamo and G. Buttazzo, "Optimal Scheduling for Fault-Tolerant and Firm Real-Time Systems," in *Proc. 5th Intl. Conf. on Real-Time Computing Systems and Applications*, Oct 1998

[58] H. Chetto and M. Chetto, "An Adaptive Scheduling Algorithm for Fault-Tolerant Real-Time Systems," *IEE Software Engineering Journal*, May 1991

[59] L. W. W. Liu, W. K. Shih, K. J. Lin, R. Bettati, and J. Y. Chung, "Imprecise Computations," *IEEE Proceedings*, Jan 1994

[60] C. C. Han, K. G. Shin, and J. Wu, "A Fault-Tolerant Scheduling Algorithm for Real-Time Periodic Tasks with Possible Software Faults," *IEEE Trans. Computers*, Mar 2003

[61] S. Kakarla and S. Katkoori, "Partial Evaluation Based Redundancy for Single Event Upset Mitigation in Combinational Circuits," in *Proc. 8th Annual Conf. on MAPLD*, Sep 2005

[62] W. W. Peterson and M. O. Rabin, "On Codes for Checking Logical Operations," *IBM Journal of Research and Development*, Apr 1959

[63] G. K. Wallace, "The JPEG Still Picture Compression Standard," *Communications of the ACM*, Apr 1991

[64] B. Sklar, *Digital Communications*, 2nd ed, Prentice Hall, 2000

[65] J. Euh, J. Chittamuru, and W. Burleson, "CORDIC Vector Interpolator for Power-Aware 3D Computer Graphics," *IEEE Workshop on Signal Processing Systems*, Oct 2002

[66] AFRL Detachment 15, "AMOS User's Manual – Maui Space Surveillance System," in *Proc. AMOS Technical Conf.*, Sep 2002

[67] G. Latif-Shabgahi, J. M. Bass, and S. Bennett, "A Taxonomy for Software Voting Algorithms Used in Safety-Critical Systems," *IEEE Trans. Reliability*, Sep 2004

[68] Xilinx Corp., "Application Note 197: Triple Modular Redundancy Design Techniques for Virtex FPGAs," version 1.0.1, Jul 2006

[69] J. L. Gersting, R. L. Nist, D. B. Roberts, and R. L. Van Valkenburg, "A Comparison of Voting Algorithms for N-Version Programming," in *Proc. 24th Annual Hawaii Intl. Conf. on System Sciences*, Jan 1991

[70] P. R. Lorczak, A. K. Caglayan, and D. E. Eckhardt, "A Theoretical Investigation of Generalized Voters for Redundant Systems," in *Proc. IEEE 19th Annual Intl. Symp. on Fault-Tolerant Computing Systems*, Jun 1989

[71] O. Cadenas and G. Megson, "Power performance with gated clocks of a pipelined Cordic Core," in *Proc. 5th Intl. Conf. on ASIC*, Oct 2003

[72] Y. Ye, K. Roy, and R. Drechsler, "Power Consumption in XOR-based Circuits," in *Proc. Asia and South Pacific Design Automation Conf.*, Jan 1999

[73] M. Guo, M. O. Ahmad, M. N. S. Swamy, and C. Wang, "FPGA Design and Implementation of a Low-Power Systolic Array-Based Adaptive Viterbi Decoder," *IEEE Trans. Circuits and Systems*, Feb 2005

[74] J. H. Anderson, F. N. Najm, and T. Tuan, "Active Leakage Power Optimization for FPGAs," in *Proc. ACM/SIGDA 12th Intl. Symp. on FPGAs*, Feb 2004

[75] M. Keating and P. Bricaud, *Reuse Methodology Manual*, 2nd ed, Kluwer Academic Publishers, Boston MA, 1999

[76] C. C. Wang and C. P. Kwan, "Low Power Technology Mapping by Hiding High-Transition Paths in Invisible Edges for LUT-Based FPGAs," in *Proc. IEEE Intl. Symp. on Circuits and Systems*, Jun 1997

[77] R. Pandey and S. Chattopadhyay, "Low Power Technology Mapping for LUT based FPGA – A Genetic Algorithm Approach," in *Proc. 16th Intl. Conf. on VLSI Design*, Jan 2003

[78] S. S. Demirsoy, A. Dempster, and I. Kale, "Transition Analysis on FPGA for Multiplier-Block Based FIR Filter Structures," in *Proc. 7th IEEE Intl. Conf. on Electronics, Circuits and Systems*, Dec 2000

[79] H. Kim and J. G. Chung, "Minimizing Switching Activity in Input Word by Offset and its Low Power Applications for FIR Filters," in *Proc. IEEE Intl. Symp. on Circuits and Systems*, May 2003

[80] A. Tiwari and K. A. Tomko, "Enhanced Reliability of Finite-State Machines in FPGA Through Efficient Fault Detection and Correction," *IEEE Trans. Reliability*, Sep 2005

[81] F. Wang, K. Ramamritham, and J. A. Stankovic, "Determining Redundancy Levels for Fault Tolerant Real-Time Systems," *IEEE Trans. Computers*, Feb 1995

[82] M. Yu and W. W. Dai, "Single-Layer Fanout Routing and Routability Analysis for Ball Grid Arrays," in *Proc. IEEE/ACM Intl. Conf. on Computer-Aided Design*, Nov 1995

[83] B. Bridgford and C. Carmichael, "A Cost/Benefit Framework for Evaluating Re-Configurable FPGA SEU Mitigation Techniques," in *Proc. 8th Annual Conf. on MAPLD*, Sep 2005

[84] H. Quinn, P. Graham, J. Krone, M. Caffrey, and S. Rezgui, "Radiation-Induced Multi-Bit Upsets in SRAM-Based FPGAs," in *Proc. Nuclear Science Radiation Effects Conf.*, Jul 2005

[85] K. Morgan, M. Caffrey, P. Graham, E. Johnson, B. Pratt, and M. Wirthlin, "SEU-Induced Persistent Error Propagation in FPGAs," *IEEE Trans. Nuclear Science*, Dec 2005

[86] C. Carmichael, B. Bridgford, G. Swift, and M. Napier, "A Triple Modular Redundancy Scheme for SEU Mitigation of Static Latch-Based FPGAs," in *Proc. 7th Annual Conf. on MAPLD*, Sep 2004

[87] A. Campbell, P. McDonald, and K. Ray, "Single Event Upset Rates in Space," *IEEE Trans. Nuclear Science*, Dec 1992

[88] C. Carmichael, E. Fuller, J. Fabula, and F. Lima, "Proton Testing of SEU Mitigation Methods for the Virtex FPGA," in *Proc. 4th Annual Conf. on MAPLD*, Sep 2001

[89] Xilinx Corp., "QPro Virtex 2.5V Radiation Hardened FPGAs," Technical Report DS028 (v1.2), Nov 2001

[90] R. Andraka, "A Survey of CORDIC Algorithms for FPGA Based Computers," in *Proc. ACM/SIGDA 6th Intl. Symp. on FPGAs*, Feb 1998

[91] J. E. Volder, "The CORDIC Trigonometric Computing Technique," *IRE Trans. Electronic Computers*, Sep 1959

[92] H. Jeong, J. Kim, and W. Cho, "Low-Power Multiplierless DCT Architecture Using Image Data Correlation," *IEEE Trans. Consumer Electronics*, Feb 2004

[93] S. Yu and E. E. Swartzlander Jr., "A Scaled DCT Architecture with the CORDIC Algorithm," *IEEE Trans. Signal Processing*, Jan 2002

[94] Y. H. Hu, "The Quantization Effects of the CORDIC Algorithm," *IEEE Trans. Signal Processing*, Apr 1992

[95] Xilinx Corp., "CORDIC v3.0," Product Specification DS249, Apr 2005

[96] W. W. Peterson, "On Checking an Adder," *IBM Journal of Research and Development*, Apr 1958

[97] M. B. Lin and A. Y. Oruc, "A Fault-Tolerant Permutation Network Modulo Arithmetic Processor," *IEEE Trans. VLSI Systems*, Sep 1994

[98] J. Coudeyras, "Radiation Testing of the Configurable Fault Tolerant Processor (CFTP) for Space-Based Applications," Master's Thesis, Naval Postgraduate School, 2005

[99] E. Johnson, M. Caffrey, P. Graham, N. Rollins, and M. Wirthlin, "Accelerator Validation of an FPGA SEU Simulator," *IEEE Trans. Nuclear Science*, Dec 2003

[100] N. Rollins and M. J. Wirthlin, "Reducing Energy in FPGA Multipliers Through Glitch Reduction," in *Proc. 8th Annual Conf. on MAPLD*, Sep 2005

[101] Atmel Corporation, "ATMEL AT40KEL040 Re-Programmable SRAM based FPGA Radiation Test Report," (website) http://www.klabs.org/ richcontent/fpga_content/atmel/at40kel04_mtr041105.pdf, Jun 2006

[102] H. Quinn, P. Graham, J. Krone, M. Caffrey, S. Rezgui, and C. Carmichael, "Radiation-Induced Multi-Bit Upsets in Xilinx SRAM-Based FPGAs," in *Proc. 8th Annual Conf. on MAPLD*, Sep 2005

[103] M. Surratt, "CFTP Development Environment Technical Manual," Naval Postgraduate School, May 2006

[104] K. R. Rao and P. Yip, *Discrete Cosine Transform – Algorithms, Advantages, and Applications*, Academic Press, 1990

[105] M.T. Sun, T.C. Chen, and A. M. Gottlieb, "VLSI Implementation of a 16x16 Discrete Cosine Transform," *IEEE Trans. Circuits and Systems*, Apr 1989

[106] S. Ramachandran, S. Srinivasan, and R. Chen, "EPLD-Based Architecture of Real Time 2D-Discrete Cosine Transform and Quantization for Image Compression," in *Proc. IEEE Intl. Symp. on Circuits and Systems*, Jun 1999

[107] Y. Yang, C. Wang, M. O. Ahmad, and M. N. S. Swamy, "An FPGA Implementation of an On-Line Radix-4 CORDIC IDCT Core," in *Proc. IEEE Intl. Symp. on Circuits and Systems*, May 2002

[108] N. Ahmed, T. Natarajan, and K. R. Rao, "Discrete Cosine Transform," *IEEE Trans. Computers*, Jan 1974

[109] M. Violante, L. Sterpone, M. Ceschia, D. Bortolato, P. Bernardi, M. Sonza Reorda, and A. Paccagnella, "Simulation-Based Analysis of SEU Effects in SRAM-Based FPGAs," *IEEE Trans. Nuclear Science*, Dec 2004

[110] K. Kota and J. R. Cavallaro, "Numerical Accuracy and Hardware Tradeoffs for CORDIC Arithmetic for Special-Purpose Processors," *IEEE Trans. Computers*, Jul 1993

INITIAL DISTRIBUTION

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Professor Herschel Loomis
   Naval Postgraduate School
   Monterey, California

4. Professor Jon Butler
   Naval Postgraduate School
   Monterey, California

5. Professor Douglas Fouts
   Naval Postgraduate School
   Monterey, California

6. Professor Sherif Michael
   Naval Postgraduate School
   Monterey, California

7. Professor Alan Ross
   Naval Postgraduate School
   Monterey, California

8. Joshua Snodgrass
   Naval Postgraduate School
   Monterey, California

9. Mindy Surratt
   Naval Research Laboratory
   Washington, DC

10. Tim Meehan
    Naval Research Laboratory
    Washington, DC

11. Dr Kenneth Clark
    Naval Research Laboratory
    Washington, DC